Modellierung von Kommunikationssystemen zum Zweck der Systemanalyse und des Systementwurfs

Dissertation

zur Erlangung des akademischen Grades
Doktoringenieur (Dr.-Ing.)

vorgelegt der
Fakultät für Informatik und Automatisierung
der
Technischen Universität Ilmenau

von
Dipl.-Inf. Johannes Klöckner
geboren am 5. März 1980 in Gelnhausen

Tag der Einreichung: 09.01.2015
Tag der wissenschaftlichen Aussprache: 25.11.2015

Gutachter: 1.) Univ.-Prof. Dr.-Ing. habil. Wolfgang Fengler
2.) Univ.-Prof. Dr.-Ing. habil. Armin Zimmermann
3.) Prof. Dr.-Ing. Gunar Schorcht

urn:nbn:de:gbv:ilm1-2015000721

Vorwort

Die vorliegende Arbeit entstand während meiner Tätigkeit als wissenschaftlicher Mitarbeiter im Fachgebiet Rechnerarchitektur und eingebettete Systeme sowie am Institut für Prozessmess- und Sensortechnik der Technischen Universität Ilmenau.

Eine besondere Prägung der Arbeit erfolgte durch die interessanten Projekte, die ich während meiner Tätigkeit bearbeiten durfte. Die zentrale Motivation entstand innerhalb des, gemeinsam mit der Mission Level Design GmbH durchgeführten, Verbundprojektes mit dem Thema „Modellbibliothek für das Kommunikationssystem FlexRay“. Die dort erarbeiteten Grundlagen und Konzepte konnten im Rahmen der Entwicklung und Realisierung des Prototypen einer Nanopositionier- und Nanomessmaschine im Transferprojekt „Nanopositionier- und Nanomessmaschine 200 (NPMM-200)“ zum Sonderforschungsbereich 622 „Nanopositionier- und Nanomessmaschinen“, sowie in dem Verbundprojekt „Anwenderorientierte Umsetzung hochleistungsfähiger IT-Konzepte in der NPMM-200“ vertieft, erweitert und angewendet werden.

An dieser Stelle möchte ich mich bei allen bedanken, die mich während der Erstellung dieser Arbeit sowohl fachlich als auch menschlich unterstützt haben. In erster Linie bedanke ich mich bei Professor Wolfgang Fengler, der mir die Tätigkeit am Fachgebiet Rechnerarchitektur und eingebettete Systeme ermöglichte und mich bei der Erstellung der Arbeit fachlich begleitete und mir stetig Freiraum gewährte. Zudem möchte ich mich bei den Professoren Gerd Jäger und Eberhard Manske für die einzigartige Möglichkeit, aktiv an der Entwicklung des Prototypen der NPMM-200 teilzunehmen, bedanken.

Des Weiteren bedanke ich mich bei den Mitarbeitern der Fachgebiete Rechnerarchitektur und eingebettete Systeme sowie Softwaresysteme/Prozessinformatik und dem Institut für Prozessmess- und Sensortechnik sowie allen beteiligten Projektpartnern. Insbesondere möchte ich mich bei meinen Kollegen Tobias Braune, Bernd Däne, Irina Kaiser, Marcus Müller und Alexander Pacholik für die sachlichen gewinnbringenden Diskussionen, die Anregungen und die aktive Unterstützung bei dieser Arbeit bedanken.

Große Unterstützung erfuhr ich zudem durch meine ehemaligen Studenten Thomas Hohmann, Normen Weller, Rene Bischof, Yixin Huang, Jens Kappert, Martin Rauh, Robin Hausdörfer, Alexey Moskalev, Oliver Brandel, Zhenhua Zhu, Thomas Amberg, Robert Boller, Bo Liu, Alexander Rothe, Stephan Simon und Sandra Timmler. Ohne sie wäre die Durchführung der Arbeit undenkbar gewesen.

Nicht zuletzt möchte ich mich bei meiner Familie für ihr unerschöpfliches Verständnis und die zusätzliche Motivation bedanken. Ohne diesen Rückhalt wäre diese Arbeit nicht möglich gewesen.

Kurzzusammenfassung

Einen wesentlichen Beitrag zu Innovationen und Weiterentwicklungen in der Automobilindustrie leisten elektronische Komponenten. Der funktionale Wachstum in den Bereichen Sicherheit, Komfort und Fahrerassistenz führt zu einer Erhöhung der Komplexität. Neben der Anzahl der Komponenten steigert sich auch der Bedarf an Kooperation und Datenaustausch. Insbesondere by-wire- und Assistenzsysteme (Hochautomatisiertes Fahren) zeichnen sich durch hohe Anforderungen in den Bereichen Zuverlässigkeit, Datenkonsistenz, Fehlertoleranz und Ausfallsicherheit aus. Die Zusammenarbeit einzelner Steuergeräte fordert von der Kommunikationsstruktur neben hohen Datenraten auch Determinismus und Echtzeitverhalten. Die Entwicklung dieser komplexen verteilten Systeme profitiert durch modellbasierte Entwurfsprozesse.

Der Nachweis von grundlegenden Systemeigenschaften mit dem Schwerpunkt Kommunikation soll bereits in frühen Entwurfsphasen mit Hilfe von ausführbaren Spezifikationen modell- und simulationsbasiert erfolgen.

In dieser Arbeit wird ein Modellierungsansatz entworfen, welcher die typischen ereignis- und zeitgesteuerten Protokolle in der Domäne Automotive adressiert. Der Fokus liegt auf den Buszugriffsverfahren. Modelle auf unterschiedlichen Abstraktionsebenen werden am Beispiel von Controller Area Network (CAN) und FlexRay definiert und realisiert. Neben der reinen Kommunikation werden die angrenzenden Themenfelder Gateway (heterogene Kopplung) und Betriebssystem berücksichtigt.

Detaillierte Modelle eignen sich zur Analyse spezifischer Protokolleigenschaften sowie zur Weiterentwicklung von Protokollfunktionen auf Modellebene. Mit abstrakteren Modellen lassen sich Leistungs- und Eigenschaftsanalysen von großen heterogenen Systemen durchführen.

Echtzeitkommunikation, vernetzte Systeme und Anwendungsfelder für modellbasierte Entwurfsprozesse finden sich auch außerhalb des Automobilbereiches. Die Anwendung wird am Beispiel der Entwicklung und Optimierung eines komplexen verteilten Systems zur Steuerung einer Nanopositionier- und Nanomessmaschine demonstriert. Innerhalb des Entwicklungsprozesses werden Entwurf, Realisierung und Leistungsbewertung bezüglich der Architektur des Gesamtsystems, der Verteilung von Funktionen und der Realisierung einzelner Komponenten sowie applikationsspezifische Kommunikationsprotokolle betrachtet.

Abstract

Major innovations and improvments in the automotive industry base on the electronic components. The growing number of functionality in the areas safety, comfort and driver assistance lead to an increase of the complexity. Not only the number of components increase. Especially to realize complex assistance systems the cooperation and data exchange gets more important. In particular, by-wire and assistance systems (highly-automated driving) have high requirements on reliability, data consistency and safety. The cooperation of single control units to realize these complex functions require not only high data rates but also determinism and real-time behavior of the communication architecture. The development of these complex distributed systems benefits from model-based design processes.

The verification and validation of system properties with a focus on communication should be possible in early design phases using model and simulation-based approaches based on executable specifications.

In this thesis, a modeling approach is developed addressing the typical event-driven and time-triggered protocols in the automotive domain. Models on different abstraction levels are defined and implemented. The Controller Area Network (CAN) and FlexRay are used as examples. Beside the communication protocols some related topics: gateway-functionality (heterogeneous communication) and operating system.

The developed detailed models are adequate for the analysis of specific protocol properties as well as the improvement of protocol functions on model level. More abstract models can be used to analyze the performance, real-time behavior and characteristics of large heterogeneous systems.

Real-time communication, distributed embedded systems and model-based design processes are not limitted to the automotive sector. Therefore the utilization of the modeling approach is demonstrated within the development and optimization of a distributed embedded system: a signal- and dataprocessing unit of a nanopositioning- and nanomeasuringmachine.

The example covers most parts of the development process. Selected topics are the design of the systemarchitecture, the distributed allocation of functionality, the realization of single components and the development of application specific communication protocols.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung und Motivation

1.1 Thematische Einordnung

In der Automobilindustrie werden zunehmend mehr elektronische Komponenten eingesetzt. Ein großes Feld für Innovationen und Weiterentwicklungen ergibt sich in den Bereichen Sicherheit und Komfort sowie Fahrerassistenz. Hierdurch wächst die Zahl der verwendeten verteilten Systeme an, zudem steigt die Komplexität und der Aspekt der Sicherheit nimmt zu. Die Grundlage bilden dafür die verbauten miteinander kommunizierenden elektronischen Komponenten. Diese Steuergeräte tauschen untereinander Daten aus und nutzen dafür als Schnittstelle spezielle Kommunikationssysteme für den Automotive-Bereich. Ein universelles Bussystem im Automobil existiert nicht, dies lässt sich auf die unterschiedlichen Anforderungen an den Datenaustausch bzw. die Kommunikation durch die einzelnen Anwendungen sowie auf ökonomische Aspekte zurückführen. Somit kommt in einem Automobil nicht nur ein Kommunikationssystem zum Einsatz sondern mehrere, und es entsteht so ein heterogenes Netzwerk. Zu den klassischen Vertretern zählen Local Interconnect Network (LIN), Media Oriented Systems Transport (MOST) und Controller Area Network (CAN).

Neuere Anwendungen und Funktionen wie beispielsweise ESP (Elektronisches Stabilitäts-Programm), Drive-by-wire, Break-by-wire und komplexe Assistenzsysteme führen zu steigenden Anforderungen an die elektrischen Systeme und die zugehörige Kommunikation. Insbesondere die x-by-wire-Systeme besitzen hohe Anforderungen in den Bereichen Zuverlässigkeit, Datenkonsistenz, Fehlertoleranz und Ausfallsicherheit. Neben hohen Datenraten wird auch eine deterministische Datenübertragung für diese Anwendungen gefordert. Insbesondere der Determinismus und das damit verbundene Echtzeitverhalten lassen sich durch den Einsatz von zeitgesteuerten Kommunikationsverfahren realisieren. Zu diesen Kommunikationsprotokollen gehören der FlexRay und das Time Triggered Protocol C (TTP/C).(Nolte, Hansson & Bello, 2005)

Die einzelnen Steuergeräte in einem Fahrzeug sind spezifische Hard-Software-Systeme deren Entwicklung zunehmend schwieriger wird. Dies begründet sich zunehmend auch durch die heterogene Kommunikationsarchitektur sowie neue Kommunikationsverfahren. Als ein relevantes Beispiel sei an dieser Stelle die Einführung des FlexRay, mit seinem zeitgesteuerten Zugriffsverfahrens, genannt. Die Entwicklung des FlexRay startete im Jahr 2000. Das Bussystem soll in Bereichen mit erhöhten Echtzeitanforderungen die Kommunikation über den CAN ersetzten. Der erste Einsatz des Bussystems für ein Teilsystem in einem Kraftfahrzeug erfolgte im Jahr 2006.

Die Komplexität der Kommunikationsarchitektur in einem Automobil steigt so weiter an und die Frage bezüglich der Migration von Teilsystemen und der Austausch von Bussystemen ist eine aktuelle Thematik. Dies gilt insbesondere für die beiden Ver-

treter CAN und FlexRay. Der CAN ist das am weitesten verbreitete Protokoll. Auf Grund der zentralen Unterschiede (Ereignisorientierung vs. Zeitsteuerung) der beiden Protokolle, ist der Übergang zwischen CAN und FlexRay besonders interessant.

Die Anpassung und Veränderung von Systemarchitekturen mit den damit verbundenen Entwurfsentscheidungen durch neue Kommunikationsprotokolle ist eine zeitlose Fragestellung. Der CAN-FD (Controller Area Network - Flexible Data Rate) ist ein aktuelles Beispiel für die Weiterentwicklung in dem Bereich der Kommunikationsprotokolle für fahrzeugtechnische Anwendungen. Eine besondere Herausforderung bei der Entwicklung der Systeme ergibt sich im Design des Gesamtsystems durch die gemeinsame Verwendung von ereignisbasierter (CAN) und zeitgesteuerter (FlexRay) Kommunikation sowie bei der Migration von Teilnetzen von CAN zu FlexRay. Das verwendete Kommunikationssystem hat einen Einfluss auf den Entwurfsprozess sowie auf das Verhalten der Anwendung auf Grund der spezifischen Protokolleigenschaften. Insbesondere bei der Verwendung von zeitgesteuerten Kommunikationssystemen fällt dem Netzwerk-Design eine besondere Rolle zu.

Die Entwicklung solcher komplexen verteilten Systeme kann durch modellbasierte Entwurfsprozesse unterstützt werden. Zum Einsatz kommen hier Toolsysteme wie MATLAB/Simulink. Die Funktionen und Eigenschaften einzelner Steuergeräte werden unter anderem mit Hardware-in-the-Loop oder Software-in-the-Loop getestet. Gesamtsystemtests werden abschließend mit Prototypen durchgeführt.

Echtzeitkommunikation, vernetzte Systeme und Anwendungsfelder für modellbasierte Entwurfsprozesse finden sich auch außerhalb des Automobilbereich. Daher lassen sich Erkenntnisse aus dem Segment Automotive auch auf andere Bereiche übertragen. Als Beispiel sei hier die Mess- und Automatisierungstechnik genannt, da ein Anwendungsbeispiel aus dieser Domäne in Kapitel 5 näher betrachtet wird. Es handelt sich dabei um ein komplexes verteiltes System zur Steuerung einer Nanopositionier- und Nanomessmaschine.

1.2 Motivation und Zielstellung

Ausgehend von diesem Stand sind die Simulation und die Analyse des Verhaltens und der Eigenschaften eines Gesamtsystems unter Verwendung von Modellen bereits während der Entwicklung der einzelnen Steuergeräte und noch vor der Realisierung von Prototypen eine zentrale Zielstellung dieser Arbeit. Der Schwerpunkt liegt dabei auf der Berücksichtigung der spezifischen Eigenschaften der Kommunikation.

In frühen Phasen des modellbasierten Entwicklungsprozesses sollen ausführbare Spezifikationen entstehen, welche die Entwicklung stark vereinfachen. So kann bereits in frühen Phasen ein Test in einem applikationsspezifischen Szenario modell- und simulationsbasiert erfolgen. Grundlegende Systemeigenschaften können so bereits in frühen Entwurfsphasen ohne Verwendung von kostenintensivem Prototyping geprüft werden. Die ausführbaren Spezifikationen können zudem zu einer Verkürzung der Entwicklungszeit beitragen. Als Nachteil von Prototypen sind die eingeschränkten Analyse- und Debug-Fähigkeiten sowie die aufwändige Erzeugung von Testszenarien im Vergleich zu simulativen Untersuchungen zu nennen. Diesen stehen natürlich im Allgemeinen schlechte Laufzeiteigenschaften und die wahrscheinlichkeitsbehaftete

Aussagekraft der Modelle gegenüber.

Zu diesem Zweck soll ein für die Domäne Automotive geeignetes Modellierungskonzept entworfen werden, welches die typischen ereignis- und zeitgesteuerten Protokolle adressiert. Die Modellierung von Kommunikationsnetzen für Automotive-Anwendungen im Umfeld des Protokolls FlexRay erfordert die Berücksichtigung von heterogenen, verteilten, eingebetteten Kommunikationsnetzen, dies schließt das Protokoll des CAN mit ein. Im Mittelpunkt der Betrachtung stehen daher die Protokolle FlexRay und CAN mit ihrem grundlegenden Unterschied im Rahmen des Buszugriffsverfahrens, welches im Hinblick auf die Modellierung eine zentrale Bedeutung besitzt.

Einen Aspekt für modellbasierte Untersuchungen spiegeln realitätsnahe Modelle wieder, welche es erlauben spezifische Protokolleigenschaften zu untersuchen oder Weiterentwicklungen von Protokollfunktionen auf der Ebene von Modellen vorzunehmen. Einen weiteren Aspekt bilden Leistungs- und Eigenschaftsanalysen auf einem abstrakteren Niveau. Diese ermöglichen die Untersuchungen von großen, komplexen heterogenen Systemen. Für die relevanten Netzwerkkomponenten werden entsprechende Modelle entwickelt.

Zur Demonstration der Allgemeingültigkeit erfolgt eine Erweiterung des entworfenen Modellierungsansatzes auf eine zusätzliche Anwendungsdomäne: Mess- und Automatisierungstechnik. Die Erprobung des Modellierungsansatzes erfolgt hierbei anhand der Entwicklung des Prototypen einer Signal- und Datenverarbeitungseinheit für eine Nanopositionier- und Nanomessmaschine. Im Rahmen des Entwurfs und der Entwicklung dieses komplexen verteilten Systems wird unter anderem die Entwicklung von eigenen angepassten Kommunikationsprotokollen notwendig, da Standards auf Grund der hohen Leistungsanforderungen nicht zur Anwendung kommen können. Innerhalb des Entwurfsprozesses wird das entwickelte Konzept angewendet. Die Übertragung der Erfahrungen und Methoden des Modellierungsansatzes haben eine positive Wirkung auf den Systementwurf. Die modellbasierte Systementwicklung wird in einem abgeschlossenen Kapitel betrachtet (Kapitel 5).

1.3 Aufbau der Arbeit

Die Arbeit gliedert sich in insgesamt sechs Kapitel inklusive der Einleitung und der Zusammenfassung und beinhaltet zudem einen ergänzenden, erweiternden und vertiefenden Anhang. Die vier Inhaltskapitel lassen sich in drei thematische Bereiche aufteilen: Grundlagen und Stand der Technik, Entwicklung eines Modellkonzeptes und Ableitung von spezifischen Modellen und Modellbibliotheken sowie Darstellung des Applikationsbeispiels Nanopositionier- und Nanomessmaschine.

Innerhalb des Kapitels 2 werden ausgewählte Grundlagen der für die Arbeit relevanten Themenbereiche dargestellt. Zu Beginn wird die Kommunikation innerhalb der Domäne Fahrzeugtechnik behandelt. Es werden aufbauend auf einer Klassifikation der Kommunikation in der Domäne ausgewählte Bussysteme vorgestellt (LIN, CAN, TTCAN, FlexRay, TTP/C) und verglichen. Anschließend werden kurz Standards zur Beschreibung von Netzwerken genannt und die Kopplung einzelner Kommunikationsnetze mit Gateways zur Erzeugung komplexer Architekturen betrachtet. Abschließend wird die Kommunikation in der Domäne Automotive anhand der Automatisierungs-

pyramide in Bezug zu der klassischen Einordnung von Techniken und Systemen in der Prozessdatenverarbeitung gesetzt.

In einem Fahrzeug werden die einzelnen Steuergeräte mit ihrer systemrelevanten Funktionalität (Anwendung) über die Bussysteme miteinander verbunden. Die Modellierung soll Aspekte der Anwendungsebene berücksichtigen, zu diesem Zweck werden ergänzend zu der Kommunikation die Betriebssystem Standards OSEK-OS und OSEK/time-OS eingeführt.

Geschlossen wird dieser Grundlagenabschnitt mit der Betrachtung von existierenden Methoden, Techniken und Tools zur Modellierung von Kommunikationsarchitekturen.

Die nächsten beiden Kapitel (Kapitel 3 und Kapitel 4) bilden den ersten Hauptteil der Arbeit, welcher sich mit der Entwicklung eines grundlegenden Modellierungsansatzes sowie der Anwendung und Verwendung dieses Ansatzes auseinandersetzt. Nach einer kurzen Betrachtung von grundlegenden Eigenschaften und Randbedingungen für die Modellierung erfolgt die Entwicklung des Modellierungsansatzes mit dem Ergebnis eines Meta-Modells für einfache und heterogene Kommunikationsarchitekturen. Aufbauend auf das entwickelte Konzept werden ausgewählte Kommunikationsprotokolle (insbesondere CAN und FlexRay) auf unterschiedlichen Abstraktionsebenen modelliert. An dieser Stelle kommen unterschiedliche Modellierungstechniken zum Einsatz. Ergänzend zur Kommunikation erfolgen die Modellierung eines variablen Gateways (Gateway-Modell-Architektur) zur Simulation komplexer Architekturen sowie die exemplarische Modellierung der Betriebssystemstandards für eine detaillierte Abbildung der Anwendungsebene.

Innerhalb des zweiten Hauptteiles der Arbeit (Kapitel 5) wird der entwickelte Modellierungsansatz auf die Domäne Mess- und Automatisierungstechnik erweitert. An dem Beispiel einer Signal- und Datenverarbeitungseinheit zur Steuerung einer Hochpräzisionspositionier und -messmaschine wird die Verwendung des Ansatzes im Rahmen der Systementwicklung demonstriert. Diese „start-from-scratch“-Entwicklung resultiert in einem reproduzierbaren Prototyp und umfasst die Entwicklung, Validierung und Realisierung eigener Kommunikationsprotokolle, Funktionskomponenten und einer leistungsfähigen Systemarchitektur.

Abschließend erfolgt in Kapitel 6 eine zusammenfassende Betrachtung der Arbeit und sinnvoller Weiterführungen.

2 Grundlagen und Stand der Technik

In diesem Abschnitt werden Grundlagen der betrachteten Themenbereiche dargestellt. Einen Schwerpunkt der Arbeit bildet die Modellierung von Kommunikationsarchitekturen in der Domäne Automotive.

Durch die Einführung von neuen Kommunikationssystemen in bestehende Systeme bzw. Architekturen ergibt sich eine Änderung der Anforderungen an den Entwurfsprozess (Nossal, 2004). Mit der Einführung von FlexRay ergaben sich somit interessante Fragestellungen im Hinblick auf die Integration der neuen Kommunikation in eine bestehende Architektur sowie die Migration von Systemen. Im Automobil existieren folglich Bussysteme mit sehr unterschiedlichen Zugriffssemantiken, welche miteinander interagieren müssen. Die Kombination von unterschiedlichen Zugriffssemantiken wirkt sich auf die resultierenden Systemeigenschaften aus.

Zunächst wird die Kommunikation in der Fahrzeugtechnik näher betrachtet. Nach einer Klassifikation der einzelnen Bussysteme werden einige ausgewählte Kommunikationsprotokolle näher betrachtet. Ausführlich werden der CAN und der FlexRay betrachtet, da sie als ein zentrales Beispiel in dieser Arbeit dienen. Der FlexRay bildet dabei das Referenzsystem für ein Kommunikationsprotokoll mit einem zeitgesteuerten Zugriffsverfahren. Der CAN als etabliertes ereignisgesteuertes Kommunikationsprotokoll bildet den Gegenpart zum FlexRay. Der Verbund von CAN und FlexRay ergibt eine heterogene Kommunikationsstruktur. Nach einer kurzen Betrachtung der Beschreibung und Spezifikation der Kommunikation in einem System wird die Kopplung von einzelnen Bussen über ein Gateway thematisiert.

Neben der Kommunikation erfolgt eine grundlegende Berücksichtigung der Anwendung und somit der Eigenschaften eines einzelnen Netzwerkknotens. Im Speziellen wird auf der Applikationsebene die Funktionalität eines Betriebssystems betrachtet. Ein entsprechender domänenbezogener Standard wird eingeführt. Abschließend erfolgt in diesem Kapitel die Betrachtung von Modellierungstechniken.

2.1 Kommunikation in der Fahrzeugtechnik

In der Automobilindustrie wird eine Vielzahl von elektronischen Komponenten zur Realisierung von Funktionen im Bereich Sicherheit und Komfort eingesetzt. Aufgrund der hohen Anforderungen an diese elektrischen Systeme arbeiten die Komponenten im Verbund und tauschen Daten untereinander aus. Die Schnittstelle zum Datenaustausch bilden speziell für dieses Segment entwickelte Kommunikationssysteme (Bussysteme). Zu diesen gehören beispielsweise LIN (Local Interconnect Network), MOST (Media Oriented Systems Transport), CAN (Controller Area Network) und als neuer Vertreter der FlexRay.(Zimmermann & Schmidgall, 2011)

Durch Anwendungen wie ESP, Drive-by-wire, Break-by-wire sowie weiterer komplexer Assistenzsysteme sind die Anforderungen in den letzten Jahren deutlich gestiegen. Dies führte im Bereich der Kommunikation zu einem Bedarf an neuen Standards. Diese Bedarfslücke, welche sich durch einen Datenaustausch mit einem hohen Datendurchsatz in Verbindung mit einer deterministischen Datenübertragung kennzeichnet, wird durch den FlexRay geschlossen.(Poledna & Kroiss, 1999; Rausch, 2008; Heinecke et al., 2002; Poledna, Ettlmayr & Novak, 2001)

Die Kommunikationsprotokolle sind in erster Linie für den Einsatz in Verbindung mit KFZ-Steuergeräten konzipiert. Diese einzelnen Steuergeräte sind spezifische Hardware-Software-Systeme, deren Entwicklung zunehmend schwieriger und durch neue Kommunikationsverfahren wie beispielsweise FlexRay wesentlich komplexer wird. Durch die sehr heterogene Kommunikationsstruktur sowie die gewachsenen Anforderungen insbesondere im Bereich Echtzeitverarbeitung wächst die Bedeutung der Kommunikation innerhalb des Automobils. Dies ergibt sich durch den Einfluss von System-Architektur, Kommunikationsstruktur, Verteilung von Funktionen sowie Wahl und Konfiguration der Kommunikation auf das Verhalten des gesamten Systems.

Durch die sehr unterschiedlichen Anforderungen wie zum Beispiel Übertragungsgeschwindigkeit, Buslänge, Fehlertoleranz, Kosten, Sicherheit u.ä. gibt es auch im Automobil keinen Universalbus, sondern eine sehr heterogene Infrastruktur (vgl. Abbildung 2.1.1).(Zimmermann & Schmidgall, 2011)

2.1.1 Das Automobil als vernetztes System

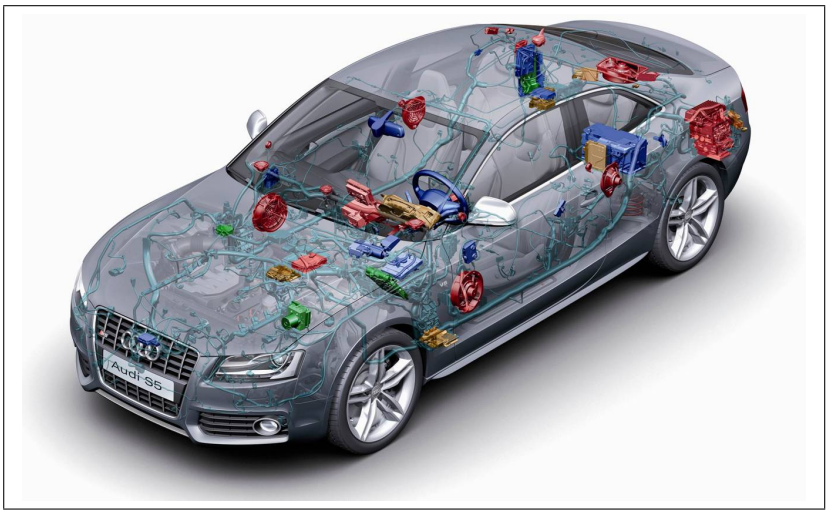


Abbildung 2.1.1: Komplexe Kommunikationsarchitektur in einem Kraftfahrzeug [1]

[1] Grafik entnommen: Webseite zur Lehrveranstaltung „Automotive Systems & Software Engineering“ (Friedrich-Alexander-Universtität Erlangen-Nürnberg), http://www4.cs.fau.de/Lehre/SS09/V_ASSE/ [22.12.2014]

Der Einsatz von Kommunikationsbussen im Automobil wurde notwendig aufgrund steigender Komplexität von Funktionen, steigender systemübergreifender Kommunikation sowie weiteren Anforderungen. Mit der Einführung von herstellerübergreifenden, vereinheitlichten Bussystemen erfolgte eine Reduktion der Produktionskosten. Damit verbunden sind weitere Vorteile wie die Integration von weiteren Sensoren, deren Mehrfachnutzung, eine Gewichtsreduzierung des Kabelbaums sowie eine Reduzierung der Anzahl und der Größe von Steckverbindern. Die Busse verringern die Anzahl von physikalischen Verbindungen. Dadurch reduzieren sie das Ausfallrisiko und führen somit zu einer Qualitätsverbesserung. Die Integration neuer und verbesserter Funktionen (Sicherheit), Erfüllung von (Abgas-) Vorschriften, Optimierung des Verbrauchs und der Ersatz mechanischer/elektrischer Systeme sind weitere positive Effekte.(Grzemba & Wense, 2005)

In Abbildung 2.1.1 ist ein Beispiel der so entstandenen komplexen Kommunikationsarchitektur in einem Fahrzeug dargestellt. Die Vielzahl an Steuergeräten und deren Kommunikationsbeziehungen bzw. -verbindungen sind erkennbar.

2.1.2 ISO/OSI-Modell

Das ISO/OSI-Modell (ISO - International Standardization Organization, OSI - Open-Systems-Interconnection) dient als Referenzmodell der Datenkommunikation und als Orientierung zur Beschreibung und Spezifikation von Kommunikationsprotokollen. Die Beschreibung eines Datenkommunikationssystems erfolgt in Form eines Schichtenmodells. Dabei wird die komplexe Gesamtaufgabe in übersichtliche, aufeinander aufbauende Funktionsbereiche (Schichten) aufgeteilt. Die folgende Darstellung beruht auf (Comer, 2004) und (ISO/IEC 7498-1:1994, 1994).

Die Dienste (Services) einer Instanz werden der jeweils darüber liegenden Schicht über Dienstzugangspunkte (SAP-Service Access Points) angeboten. Eine Instanz kommuniziert jeweils mit der Partner-Instanz. Der Austausch von Daten erfolgt durch Inanspruchnahme von Diensten in der unterliegenden Schicht.

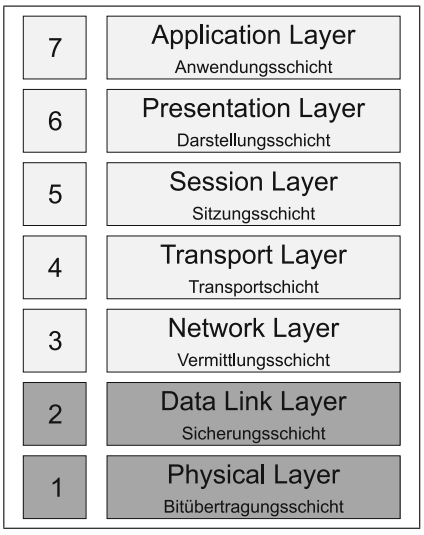


Abbildung 2.1.2: ISO-OSI-Referenzmodell

In dem Referenzmodell werden wie in Abbildung 2.1.2 dargestellt sieben Schichten definiert:

Schicht 1: Physikalische Schicht (Physical Layer) Die Übertragung von Bits über den Kommunikationskanal erfolgt in der *Physikalischen Schicht* (Bitübertragungsschicht), somit erfolgt die Spezifikation der Übertragung von Bitströmen. Die wichtigsten Parameter sind das physikalische Übertragungsmedium sowie die elektrischen, mechanischen und prozeduralen Schnittstellen. Hierzu zählt auch die Festlegung von Kabeln und Konnektoren. Wichtige Eigenschaften zur Übertragung von Bits sind beispielsweise die Zeitsynchronisation (Synchronisation von Bitzeiten), die Festlegung der Bitrepräsentation auf dem Übertragungsmedium (Kodierung) und Modulation.

Schicht 2: Verbindungsschicht (Data Link Layer) Die Datenpakete aus den übergeordneten Schichten werden in der Verbindungsschicht in Rahmen aufgeteilt und mit Zusatzinformationen versehen (Framing). Mit diesen Zusatzinformationen werden die Rahmenbegrenzungen realisiert und es erfolgen die Adressierung auf einem niedrigen Niveau sowie ein einfaches Multiplexing/Demultiplexing. Neben der Synchronisation der Rahmen werden in der Verbindungsschicht das Buszugriffsverfahren (Medium Access Control) festgelegt und die Fehlersicherung und die Flusskontrolle vorgenommen.

Schicht 3: Netzwerkschicht (Network Layer) Die zentrale Aufgabe der Netzwerkschicht (Vermittlungsschicht) ist in großen Netzen einen Weg vom Sender zum Empfänger, oftmals über heterogene Teilnetze, zu ermitteln (Routing), d.h. es erfolgt ein Datentransfer zwischen den Endpunkten eines Netzwerkes. Die Vermittlung der Pakete erfolgt anhand statischer oder dynamischer Routing-Tabellen in den Verbindungsknoten (Router), welche einzelne Netzsegmente verbinden. Zu den Aufgaben der Übertragung von Paketen bzw. Datagrammen gehören der Verbindungsaufbau, die Wegewahl (Routing), Fragmentierung von Paketen, die Vermittlung sowie die Betriebsmittelverwaltung.

Schicht 4: Transportschicht (Transport Layer) Der zuverlässige Transfer der Daten zwischen den Endpunkten eines Netzwerkes ist die wesentliche Aufgabe der Transportschicht. Die Transportschicht ist für die Fehlerbehandlung (Wiederherstellung einer Verbindung bei Fehlern im darunter liegenden Netz), Flusskontrolle, Namensgebung und Adressierung zuständig.

Schicht 5: Sitzungsschicht (Session Layer) Die Sitzungsschicht stellt Dienste bereit, welche einen Datenaustausch zwischen Anwendungen und Endsystemen ermöglicht. Die Aufgaben umfassen den Auf- und Abbau von Verbindungen sowie deren Unterhalt. Es können beispielsweise verschiedenen Prozesse synchronisiert werden, um eine korrekte Datenübertragung zu ermöglichen.

Schicht 6: Darstellungsschicht (Presentation Layer) Die Daten werden in heterogenen Kommunikationssystemen mit einer unterschiedlichen Syntax und Semantik übertragen. Die Darstellungsschicht ist dafür verantwortlich, Vereinbarungen über den verwendeten Zeichensatz und die Codierung, Verschlüsselung

oder Komprimierung der Daten zwischen den kommunizierenden Partnern zu treffen.

Schicht 7: Anwendungsschicht (Application Layer) Die Anwendungs- bzw. Verarbeitungsschicht stellt Funktionen bereit, die es dem Anwendungsprogramm ermöglichen, (orts)transparent auf das Kommunikationssystem zuzugreifen.

Die Bussysteme im Bereich der Prozessdatenverarbeitung und auch die Bussysteme im Automobil benutzen zumeist lediglich die ersten beiden Schichten des ISO/OSI-Referenzmodells, da die Dienste der weiteren Schichten zumeist nicht benötigt werden oder von zusätzlichen speziellen Protokollen übernommen werden. Hierzu zählen beispielsweise Transportprotokolle wie FlexRay TP (ISO 10681-2) oder ISO TP (ISO 15765-2) (Zimmermann & Schmidgall, 2011).

2.1.3 Klassifikation von automotive Bussystemen

Die Society of Automotive Engineers (SAE) hat die Datenkommunikation in Kraftfahrzeugen in drei Klassen (A, B und C) unterteilt. Die Aufteilung erfolgt im Wesentlichen anhand der Bandbreite und dem Anwendungsbereich. Diese Einteilung wird mittlerweile um weitere Klassen erweitert (Grzemba & Wense, 2005; Lupini, 2001, 2003). In Abbildung 2.1.3 sind zwei dieser weiteren Klassen zu erkennen: sicherheitsrelevante Systeme und Multi-Media-Systeme (vgl. Grzemba & Wense, 2005). In (Zimmermann & Schmidgall, 2011) wird für den Anwendungsbereich Werkstatt- und Abgastester eine weitere Klasse (Diagnose) genannt. Lupini (2003) befasst sich ausführlich mit der Charakterisierung der Bussysteme im Automobil und unterscheidet insgesamt neun Kategorien (Class A; Class B; Class C; Emissions Diagnostics; Mobile Media: Low Speed, High Speed, Wireless; Safetybus; Drive-by-Wire).

In der Abbildung 2.1.3 sind nach Grzemba und Wense (2005) die einzelnen Klassen horizontal angeordnet, die vertikale Ordnung ergibt sich durch die Übertragungsrate der einzelnen Bussysteme.

Die Klassen werden folgendermaßen definiert. Unter Klasse A fällt die Kommunikation zwischen Steuergeräten und deren Sensoren, Aktoren und Bedienelementen. In diesen Sensor-Aktor-Netzwerken sind die Fehlerfreiheit und die Übertragungsgeschwindigkeit (Datenrate) von geringer Bedeutung und es genügen geringe Datenraten mit bis zu 10 kBit/s. Zu dieser SAE-Klasse gehört beispielsweise der LIN-Bus.

Die Kommunikationssysteme der Klasse B verbinden die Steuergeräte aus dem Karosseriebereich (auch Komfortbereich genannt), hierzu gehören beispielsweise Tür- oder Klimasteuergeräte.Die Bussysteme sind im Allgemeinen nicht echtzeitfähig und eigenen sich nicht für sicherheitskritische Anwendungen. Die Übertragungsraten liegen zwischen 10kBit/s und 125 kBit/s. Der Low-Speed-CAN (CAN-B oder auch Komfort-CAN) ist ein typischer Vertreter dieser Klasse.

Die Eigenschaften Datenrate (125 kBit/s bis 1Mbit/s) und Fehlertoleranz sind in der Klasse C ein wichtiges Kriterium. Zu dieser Klasse zählen im Automobil die Kommunikationssysteme im Bereich des Antriebs. Der als CAN-C oder auch Antriebs-CAN bezeichnete High-Speed-CAN kommt zum Einsatz und verbindet zum Beispiel

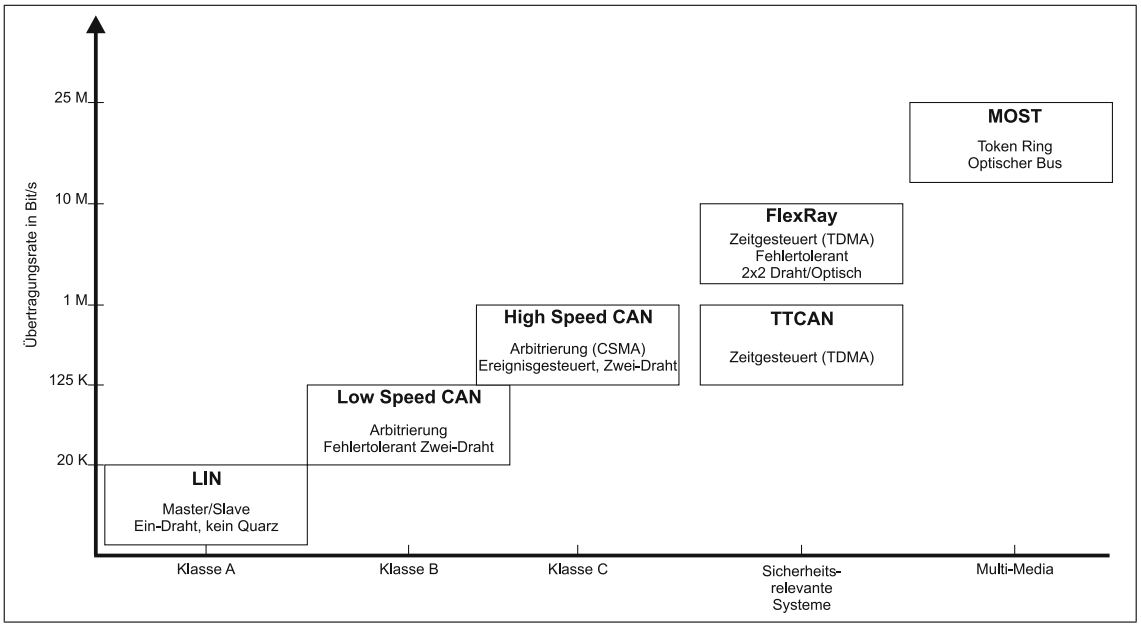


Abbildung 2.1.3: Klassifikation von Kommunikationsanforderungen der Society of Automotive Engineers (SAE) (Grzemba & Wense, 2005)

das Motorsteuergerät mit dem Getriebe- und dem Bremsregelungs-Steuergerät. Der CAN-C verwendet eine Brutto-Datenrate von maximal 500 kBit/s (zulässig wären beim High-Speed-CAN 1 MBit/s).

Die Kommunikationssysteme der Klasse *sicherheitsrelevante Funktionen* beinhalten eine Steigerung der Anforderungen bezüglich der Eigenschaften Datenrate (1 MBit/s bis 10 MBit/s), Determinismus, Echtzeitfähigkeit und Fehlertoleranz. Für diese Klasse findet man auch weitere Bezeichnungen wie „Klasse D“ (Zimmermann & Schmidgall, 2011) oder „Drive-by-Wire“ (Lupini, 2003). Dieser Klasse der *zeitgesteuerten redundanten Systeme* wird der FlexRay zugeordnet.

In der Klasse der *Multimedia-Systeme* („Mobile Media“ (Lupini, 2003)) mit den höchsten Anforderungen an die Datenrate (250 kBit/s bis mehr als 100 Mbit/s) und eingeschränkten Anforderungen an die Echtzeitfähigkeit findet sich beispielsweise der MOST-Bus für eine echtzeitfähige Audio- und Video-Übertragung.

2.1.4 Darstellung ausgewählter Bussysteme

In Bussystemen können mehrere Kommunikationspartner, welche als Knoten bezeichnet werden, über eine Sammelleitung kommunizieren. Die durch die Sammelleitung hergestellte Kommunikationsstruktur wird auch als (Kommunikations-) Cluster bezeichnet.(Schnell & Wiedemann, 2006)

In Protokollen wird die Art der Übertragung, die Adressierung innerhalb des Bussystems und die Vergabe der Buskontrolle zur Kollisionsvermeidung geregelt. Die Vergabe der Buskontrolle (Arbitrierung) kann zentral durch einen Bus-Arbiter oder dezentral durch die beteiligten Knoten erfolgen. Der Mechanismus zur Vergabe der Buskontrolle hat einen Einfluss auf die Kommunikation und somit auf die Eigenschaften und das Verhalten des Gesamtsystems.(Reissenweber, 2009)

Bussysteme können in Systembusse, Peripheriebusse und Feld- oder Prozessbusse eingeteilt werden. Systembusse verbinden die internen Komponenten eines Mikrorechnersystems und sorgen für eine möglichst schnelle Datenübertragung zwischen Prozessor, Speicher und Ein- und Ausgabeeinheiten. Zu den Peripheriebussen, welche zur Anbindung von externer Peripherie dienen, zählen beispielsweise I^2C (Inter-Integrated Circuit) und SPI (Serial Peripheral Interface). Zur Verbindung von mehreren Mikrorechnersystemen kommen Feldbusse zum Einsatz. Diese haben gesteigerte Anforderungen bezüglich Zeitverhalten, Robustheit, Sicherheit u.ä. und sind im Weiteren Gegenstand der Betrachtung.(Schnell & Wiedemann, 2006; Reissenweber, 2009)

Der Einsatz von Bussystemen in der Prozessdatenverarbeitung umfasst die unterschiedlichsten Anwendungsdomänen z.B. Automatisierung, Industrieroboter, Industrie-, Fabrik- und Haustechnik sowie insbesondere Steuer- und Regelungsaufgaben in Fahrzeugen. Neben der hier betrachteten Kommunikation in der Automobiltechnik gilt gleiches auch für Flug- und Schifffahrtstechnik.

In den unterschiedlichen Bereichen existieren sehr unterschiedliche Anforderungen an die Kommunikation und so gibt es eine Vielzahl von Feldbussen je nach Anwendungsbereich und Hersteller. Nachfolgende Auflistung gibt auszugsweise einige dieser Feldbusse an (Gruhler, 2001):

Aktor Sensor Interface Der ASI-Bus entstand als eine einfache Schnittstelle für binäre Feldgeräte aus einem Verbundprojekt.

Process Field Bus Der Profi-Bus ist ein Feldbus der von verschiedenen Firmen und Hochschulen in einem Verbundprojekt entwickelt wurde. Den Profi-Bus gibt es in verschiedenen Ausprägungen.

Controller Area Network Der CAN-Bus ist ein von Bosch und Intel entwickelter Feldbus für die Zusammenschaltung von Mikroprozessoren, Aktoren und Sensoren in Fahrzeugen.

Local Interconnect Network Der LIN ist ein low-cost Feldbus für die Kommunikation in Fahrzeugen mit geringen Anforderungen.

FlexRay Ein Feldbussystem für den Automobilbereich.

P-NET-Bus Ein für Anwender lizenzfreier Feldbus der dänischen Firma PROCESDATA.

Interbus S Ein Aktor/Sensor-Bus aus der Entwicklung eines Firmenverbundes (u.a. Phönix Kontakt).

DIN-Meßbus Durch einen DIN-Ausschuss unter Mitarbeit von Messgeräteherstellern und der Physikalisch-Technischen Bundesanstalt genormter Bus zur Datenübermittlung im Bereich Mess- und Prüftechnik.

TTP/C TTP/C ist ein auf dem Time Triggered Protocol (TTP) basierender Feldbus. TTP ist ein zeitgesteuertes Kommunikationsprotokoll für verteilte und fehlertolerante Echtzeitsysteme.

Aufgrund der Vielzahl von Anwendungsdomänen, Herstellern, Konsortien u.ä. ließe sich diese Liste noch weiterführen. Einzelne Bussysteme haben sich in unterschiedlichen Anwendungsbereichen durchgesetzt. Ein gutes Beispiel hierfür ist der CAN, der ursprünglich als Bussystem für den Automobilbereich entwickelt wurde, aber mittlerweile in einer Vielzahl von unterschiedlichen Domänen zum Einsatz kommt. Aufgrund der unterschiedlichen Umgebungsanforderungen gibt es bei der jeweiligen Verbindungstechnik in der Kraftfahrzeugtechnik, Schifffahrtstechnik[2] oder auch Industrieanlagen Variationen.

2.1.4.1 LIN - Local Interconnect Network

Der LIN-Bus (LIN Consortium, 2010) ist ein weiteres Kommunikationssystem welches im Automobilbereich zum Einsatz kommt und Übertragungsraten zwischen 1 kBit/s und 20 kBit/s ermöglicht. Konzipiert ist das Bussystem als Subbus des CAN und findet im unteren Kosten- und Leistungsbereich Verwendung. In der LIN-Bus Spezifikation werden die Schichten 1, 2, 4 und 7 des ISO/OSI-Referenzmodells adressiert. (Grzemba & Wense, 2005)

Ein LIN-System besteht aus einem Master und mehreren Slaves. Der LIN wird zumeist als Subbus des CAN eingesetzt. In diesem Fall ist der Masterknoten gleichzeitig Bestandteil eines CAN-Netzes. Der Master initiiert und steuert die Kommunikation. Zu diesem Zweck werden zwei Tasks spezifiziert, ein Master-Task und ein Slave-Task. Der Master-Knoten enthält beide Tasks während ein Slave lediglich den Slave-Task enthält. Der Master-Task steuert, durch das Senden von Botschaften-Headern, die Kommunikation und der Slave-Task ist verantwortlich für das Senden von Datenbotschaften, als Reaktion auf einen Botschaften-Header. Header und Response bilden dabei eine LIN-Nachricht. Eine ausführlichere Betrachtung des LIN findet sich im Anhang A.2.1.(Grzemba & Wense, 2005)

2.1.4.2 CAN - Controller Area Network

2.1.4.2.1 Allgemein Das CAN-Protokoll (Controller Area Network) ist eine serielles, bitstromorientiertes Bussystem, welches ursprünglich für die interne Kommunikation bei Kraftfahrzeugen entwickelt wurde (Robert Bosch GmbH, 1991). Auf Grund der Eigenschaften des Bussystems haben sich weitere Anwendungsbereiche erschlossen, wie beispielsweise Automatisierungstechnik, Antriebstechnik und Gebäudeleittechnik (Etschberger, 2002). Die nachfolgende Darstellung des Bussystems basiert dabei auf der Spezifikation (Robert Bosch GmbH, 1991) sowie (Zimmermann & Schmidgall, 2011; Etschberger, 2002)

Alle Knoten im System sind in Folge des verwendeten Multi-Master-Zugriffsverfahrens gleichberechtigt. Der Datenaustausch erfolgt in dem nachrichtenorientierten Protokoll ereignisgesteuert unter Verwendung von Prioritäten.

Innerhalb einer Nachricht ist der Transfer von einer geringen Anzahl von Nutzdaten möglich (maximal 8 $Byte$). Die maximale Datenrate beträgt 1 $MBit/s$. Der

[2] Böning Automationstechnologie, Katalog 2014 - Schiffsautomation, `https://www.boening.com/fileadmin/user_upload/catalogue/2014/Boening-Katalog-Schiffsautomation-DE-Web.pdf` [08.01.2015]

Datenaustausch erfolgt mit einer hohen Sicherheit und geringer Latenz. Einschränkungen entstehen hier allerdings durch die mögliche Blockierung von Nachrichten mit niedrigen Prioritäten. Bei harten Anforderungen an die Echtzeit können sich durch die Prioritätssteuerung und Nachrichtenblockierung Probleme ergeben.

Innerhalb des CAN-Standards werden, wie bei vielen anderen Feldbussystemen lediglich die untersten beiden Schichten des ISO/OSI-Referenzmodells beschrieben.

2.1.4.2.2 Busstruktur Eingesetzt wird der CAN primär in einer Linientopologie (siehe Abbildung 2.1.4), allerdings sind auch Punkt-zu-Punkt Verbindungen und Sternstrukturen möglich. Die Ausdehnung des Bussystems ist dabei abhängig von der verwendeten Bitrate. Für die Busausdehnung gilt der folgende Zusammenhang: $Busausdehnung \leq 40..50\ m \times \frac{1\ MBit/s}{Bitrate}$ (Zimmermann & Schmidgall, 2011).

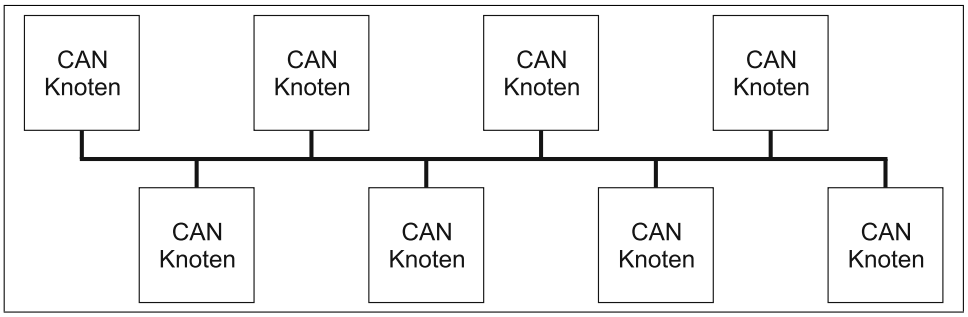


Abbildung 2.1.4: Busstruktur CAN

Als Busmedium wird eine verdrillte Zwei-Draht Leitung verwendet. Bei Systemen mit geringen Datenraten ($33 - 83\ kbit/s$) ist auch eine Ein-Drahtleitung möglich (SAE J2411). Bei Verwendung entsprechender Treiberbausteine können optische Medien verwendet werden. Auf dem Medium werden zwei Buspegel unterschieden: ein rezessiver Pegel (entspricht dem Bitwert 1) und ein dominanter Pegel (entspricht Bitwert 0).

Zur Übertragung wird die NRZ-Kodierung (non return to zero) verwendet. Zur Vermeidung von Problemen bei der Bit-Synchronisation durch längere Phasen ohne Bitwechsel, wird nach fünf gleichwertigen Bits automatisch ein invertiertes Stuff-Bit in den Bitstrom eingefügt, welches beim Empfänger wieder entfernt wird.

2.1.4.2.3 Buszugriffsverfahren Die Steuerung des Zugriffs auf den Bus erfolgt mit Hilfe eines CSMA/CA-Verfahren (Carrier Sense Multiple Access with Collision Avoidance) in Verbindung mit einer Priorisierung der Nachrichten. Jede Nachricht besitzt einen eindeutigen Identifier, der Inhalt und Priorität der Nachricht festlegt. Je niedriger der Identifier desto höher ist die Priorität der Nachricht.

Der aktuelle Buszustand wird bitweise von allen Knoten abgehört, diese können so die gesendete Nachricht empfangen. Findet kein Verkehr auf dem Bus statt, so können mehrere Teilnehmer einen Sendeversuch unternehmen. Hierdurch entsteht kein Datenverlust. Durch die Arbitrierung setzt sich nur die Nachricht mit der höchsten

Priorität durch, die anderen Teilnehmer mit Sendewunsch brechen das Senden ab. Ein Knoten wiederholt das abgebrochene Senden, sobald der Bus wieder frei ist.

Arbitrierung Das Beispiel in Tabelle 2.1 soll die Arbitrierung verdeutlichen. In einem System bestehend aus drei Knoten A, B und C haben die beiden Knoten A und B jeweils einen Sendewunsch. Knoten A möchte eine Nachricht mit dem Identifier 0x19A und Knoten B eine Nachricht mit dem Identifier 0x196 senden. Aufgrund des niedrigeren Identifier hat die Nachricht von Knoten B die höhere Priorität.

Tabelle 2.1: Beispiel für die Arbitrierung beim CAN-Bus

Bit-Nr.	1	2	3	4	5	6	7	8	9	10	11
A: Sendewunsch mit Identifier 0x19A	**0**	**0**	**1**	**1**	**0**	**0**	**1**	*1*	*0*	*1*	*0*
B: Sendewunsch mit Identifier 0x196	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**
C: kein Sendewunsch (Buszustand)	**0**	**0**	**1**	**1**	**0**	**0**	**1**	**0**	**1**	**1**	**0**

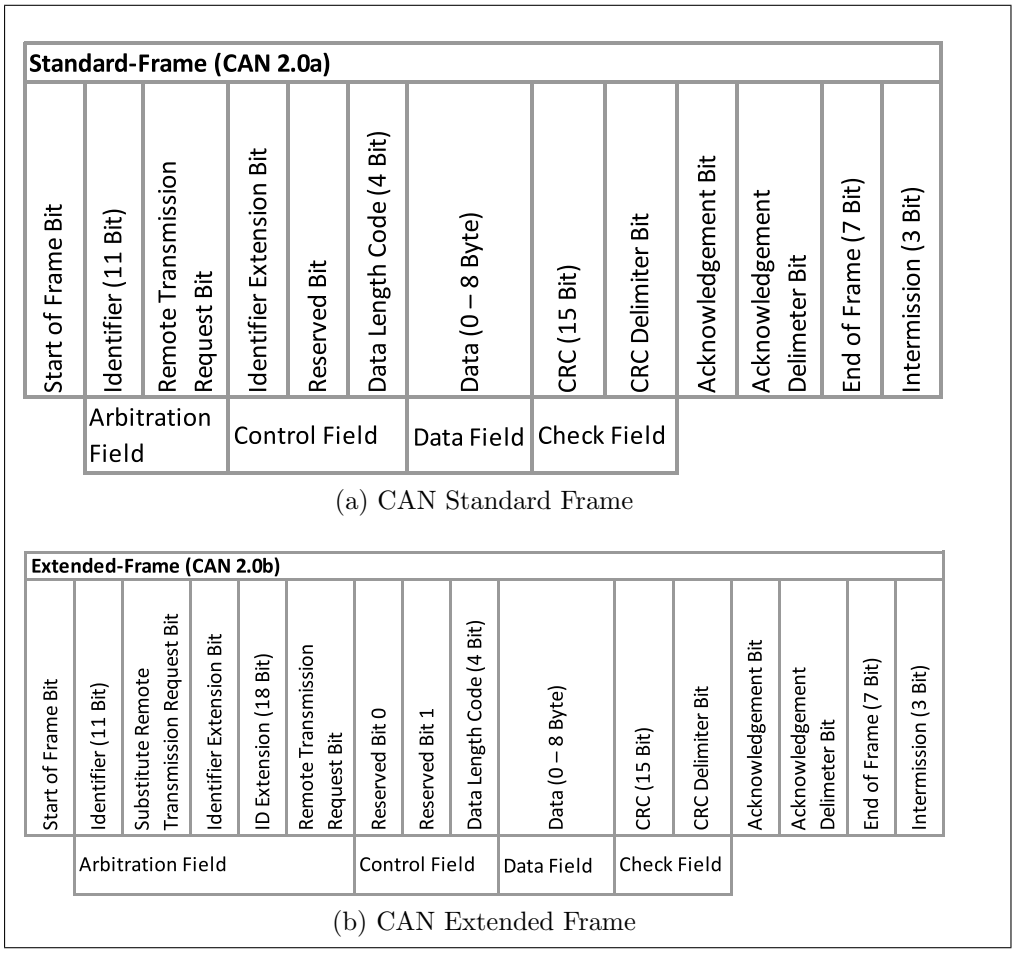
Knoten A und Knoten B beginnen gleichzeitig mit dem Senden ihrer Nachricht. Beim Senden des Identifier prüfen die Knoten jeweils ob gesendeter und empfangener Bitwert identisch sind. Dies gilt bis zu Bit Nummer acht, hier erkennt Knoten A, dass der Pegel auf dem Bus nicht dem gesendeten entspricht und bricht daraufhin das Senden ab. Somit wird die Nachricht von Knoten B auf dem Bus übertragen.

Nachrichtenformat Innerhalb des Protokolls werden vier unterschiedliche Nachrichtentypen unterschieden:

- Data-Frame
- Remote-Frame
- Error-Frame
- Overload-Frame

Das CAN Data-Frame dient zur Datenübertragung und liegt in zwei Ausprägungen vor: als CAN Standard Data-Frame (CAN 2.0A) und als CAN Extended Data-Frame (CAN 2.0B). Das Standard Data-Frame beinhaltet einen 11 Bit Identifier. Im Gegensatz dazu steht im Extended Data-Frame ein 29 Bit Identifier zur Verfügung. Der detaillierte Aufbau der Frames ist den Abbildungen 2.1.5 zu entnehmen.

Das Remote-Frame dient zur Anforderung von Data-Frames anderer Teilnehmer. Der Frameaufbau ist identisch mit dem Aufbau eines Data-Frame, mit der Ausnahme eines rezessiven RTR-Bit (Remote Transmission Request). Mit dem Error-Frame kann innerhalb des Bussystems ein Fehler signalisiert werden. Fehlerquellen sind beispielsweise eine fehlerhafte CRC oder Fehler während des Bitstuffing. Mit dem Overload-Frame signalisiert ein Teilnehmer, dass er zur Verarbeitung einer weiteren Nachricht noch nicht bereit ist.



(a) CAN Standard Frame

(b) CAN Extended Frame

Abbildung 2.1.5: CAN Daten Frame (vgl. Etschberger, 2002)

Echtzeitverhalten Für eine einzelne Nachricht lässt sich eine worst-case Übertragungszeit angeben. In der Literatur werden in (Robert Bosch GmbH, 2007) für ein Standard-Frame 130 Bitzeiten und für ein Extended-Frame 150 Bitzeiten angegeben. In (K. Tindell & Burns, 1994) werden in die Betrachtung der Verlängerung durch Bitstuffing die Stuffbits selbst auch betrachtet, so finden sich hier 135 Bitzeiten.

Aufgrund der Arbitrierung können Verzögerungen lediglich stochastisch angegeben werden. Daher gilt der CAN-Bus insbesondere bei hohen Buslasten als nicht echtzeitfähig oder nur mit Einschränkungen. Im praktischen Einsatz wird daher meist die maximale Buslast beschränkt (z.B. auf 50%), um so die Nachrichtenübertragung sicherzustellen.

2.1.4.3 TTCAN - Time Triggered CAN

Aufgrund der eingeschränkten Echtzeitfähigkeit des CAN-Bus gibt es mit dem Time-Triggered CAN (TTCAN - ISO 11898-4:2004) (Leen & Heffernan, 2002; Führer, Müller, Dieterle, Hartwich & Hugel, 2000; Zimmermann & Schmidgall, 2011; Etschberger, 2002) eine Erweiterung für den CAN-Bus. Über das CAN-Protokoll wird ein zyklusbasiertes zeitgesteuertes Zugriffsverfahren gelegt, welches softwareseitig realisiert wird

und Übertragungszeiten garantiert.

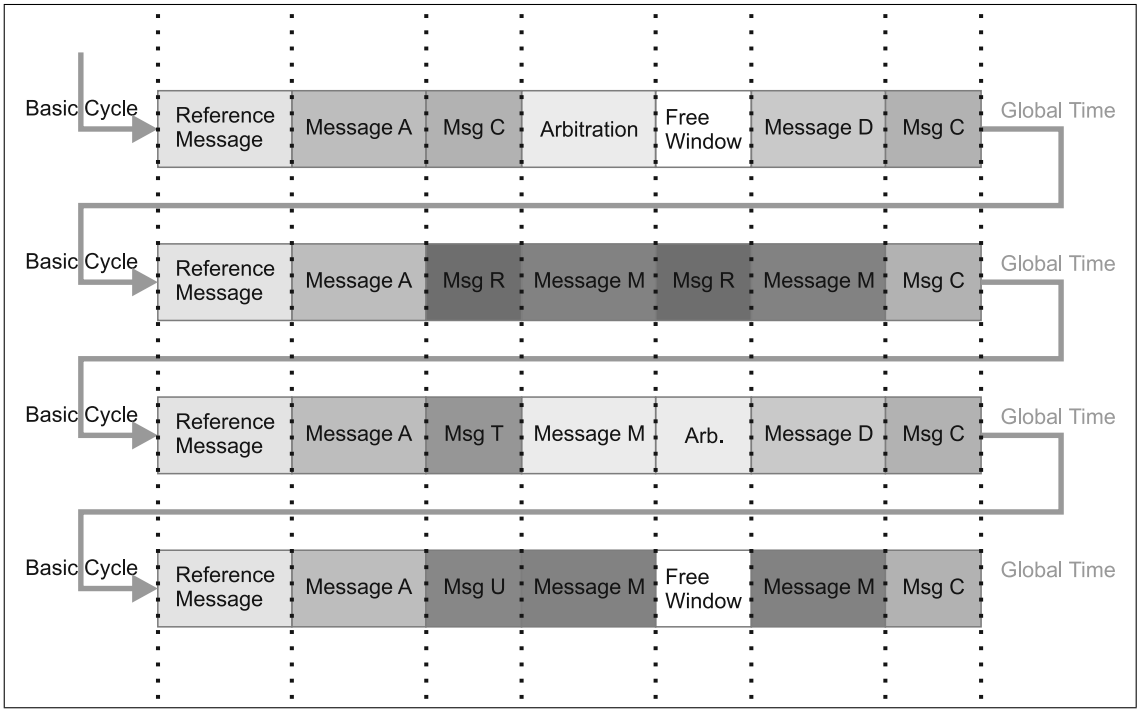


Abbildung 2.1.6: TTCAN Kommunikationszyklus (Führer et al., 2000)

Ein Knoten im System wird als Time-Master definiert und mit dessen Referenznachricht beginnt ein jeder Zyklus (Basic Cycle). Auf diese Nachricht synchronisieren sich alle Teilnehmer. Der Grundzyklus beinhaltet eine variable Anzahl an Zeitfenstern (Time Windows) mit unterschiedlichen Typen: in einem *exclusiv window* darf eine spezifische Nachricht gesendet werden, in einem *arbitration window* erfolgt der Zugriff nach dem CSMA/CA-Verfahren, ein *free window* dient für spätere Erweiterungen ohne Zyklusanpassungen. Grundzyklen in unterschiedlichen Konfigurationen können zu einem Systemzyklus zusammengefasst werden. Für die Kommunikation ist eine Zeitsynchronisation notwendig, diese beinhaltet eine Offset-Korrektur (Referenznachricht) und ein Verfahren zur Driftkorrektur (Übertragung der Systemzeit). In Abbildung 2.1.6 ist ein Beispiel für einen Systemzyklus dargestellt.

2.1.4.4 FlexRay

2.1.4.4.1 Allgemein Im Jahr 2000 wurde das FlexRay Konsortium gegründet. Die Zielstellung für die Gruppe war die Entwicklung eines neuen, schnellen und leistungsfähigen Kommunikationssystems, das die echzeitfähige Kommunikation für sicherheitsrelevante Systeme innerhalb des Automobils ermöglicht. (Rausch, 2008) Die allgemeinen Eigenschaften des Bussystems leiten sich dabei direkt aus der Zielsetzung ab:

- Bruttodatenrate von 2 x 10 Mbit/s
- Synchronisierte Zeitbasis
- Skalierbare Redundanz
- Bekannte Nachrichtenlaufzeit mit garantierter Varianz
- Flexibilität

Im Jahr 2010 hat das Konsortium seine Arbeit eingestellt und die aktuelle Spezifikation einem Standardisierungsgremium übergeben. Die Webpräsenz des Konsortiums wird weiterhin betreut, über diese sind die Protokollspezifikationen offengelegt. Zur Verfügung gestellt werden die Versionen 3.0.1 und 2.1 Rev. A. Die nachfolgende Beschreibung bezieht sich im Wesentlichen auf die offengelegte Spezifikation 2.1 Rev. A (FlexRay Consortium, 2005a, 2005b) sowie (Rausch, 2008; Heinecke et al., 2002; Zimmermann & Schmidgall, 2011).

FlexRay-Schichtenmodell Bei der Spezifikation der FlexRay-Protokollfunktionen wird sich auf die untersten beiden Schichten des ISO/OSI-Referenzmodells beschränkt (vgl. Abbildung 2.1.7). Dies begründet sich durch die Effizienz des Protokolls und der damit verbundenen Reduktion auf den erforderlichen Funktionsumfang.

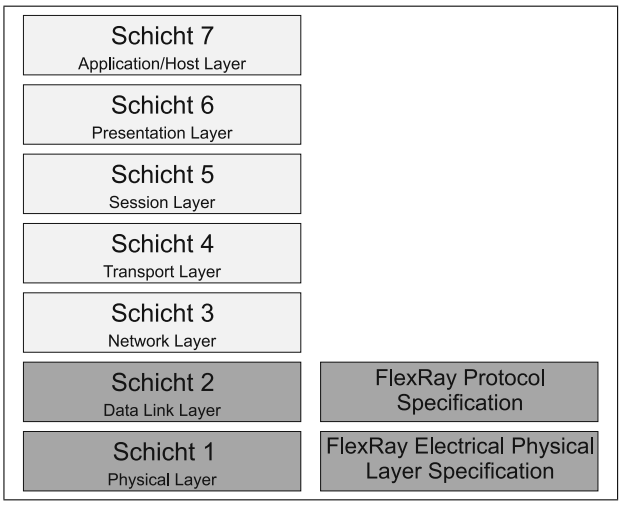


Abbildung 2.1.7: FlexRay Schichtenmodell (vgl. Rausch, 2008)

Für die beiden Schichten existieren separate Spezifikationen. Die unterste Schicht, die Physikalische Schicht, wird durch die FlexRay Electrical Physical Layer Specification abgedeckt. Die zentrale Protokollfunktionalität ist in der Schicht 2 (Verbindungsschicht) enthalten und wird durch die FlexRay Protocol Specification festgelegt.

Busstruktur Eine Besonderheit beim FlexRay ist, dass zwei getrennte Kanäle (Kanal A und Kanal B) nutzbar sind. Der FlexRay ist ein logischer Bus, welcher auf physikalischer Ebene unterschiedliche Topologien unterstützt:

- Punkt-zu-Punkt
- Bus
- passiver Stern
- aktiver Stern
- gemischte Topologien
- ein- oder zweikanalige Topologien

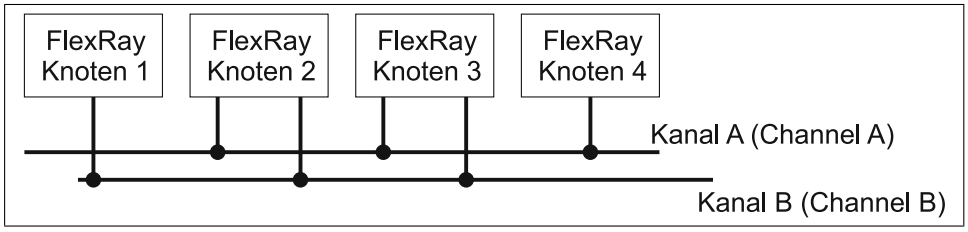


Abbildung 2.1.8: Busstruktur FlexRay (vgl. Rausch, 2008)

Ein Busknoten selbst kann wie in Abbildung 2.1.8 an einem oder beiden Kanälen angeschlossen sein und besteht aus einem Host, einem FlexRay-Controller (Realisierung FlexRay Protocol - Schicht 2) sowie zwei Bus-Treibern, welche die physikalische Verbindung zum Übertragungsmedium herstellen (Bitstrom-orientierte Übertragung). Ein FlexRay Kommunikationssystem wird als FlexRay-Cluster bezeichnet.

2.1.4.4.2 Buszugriffsverfahren Das Zugriffsverfahren des FlexRay basiert auf einem Kommunikationszyklus, der über eine gemeinsame Zeitbasis der Knoten realisiert wird.

Zeithierarchie Die globale Zeit ist hierarchisch aufgebaut. Die oberste Ebene bildet der Zyklus, dieser baut sich aus Macroticks auf. Ein einzelner Macrotick ist in allen Knoten eines Kommunikationsclusters gleich groß und bildet ebenso wie der Zyklus eine globale Zeiteinheit. Aus diesen leiten sich Slots und Minislots ab, welche zur Steuerung der Kommunikation dienen.

Die lokale Zeiteinheit Microticks bestimmt einen Macrotick und wird aus einem knotenlokalen Taktgeber (Oszillator) gewonnen. Microticks in unterschiedlichen Knoten können eine unterschiedliche Zeitdauer besitzen. Abweichungen zwischen den Knoten werden durch eine Uhrensynchronisation (Offset- und Frequenzkorrektur) korrigiert. Somit ist die Anzahl der Macroticks in einem Zyklus immer gleich, während die Anzahl der Microticks in einem Macrotick variieren kann.

Kommunikationszyklus Der FlexRay Kommunikationszyklus setzt sich aus vier Segmenten zusammen (Abbildung 2.1.9): Statisches Segment, Dynamisches Segment,

Symbol Window und Network Idle Time (NIT). Für die Kommunikation sind die beiden ersten Segmente relevant. Die NIT wird für die Berechnung und Durchführung der Uhrensynchronisation verwendet.

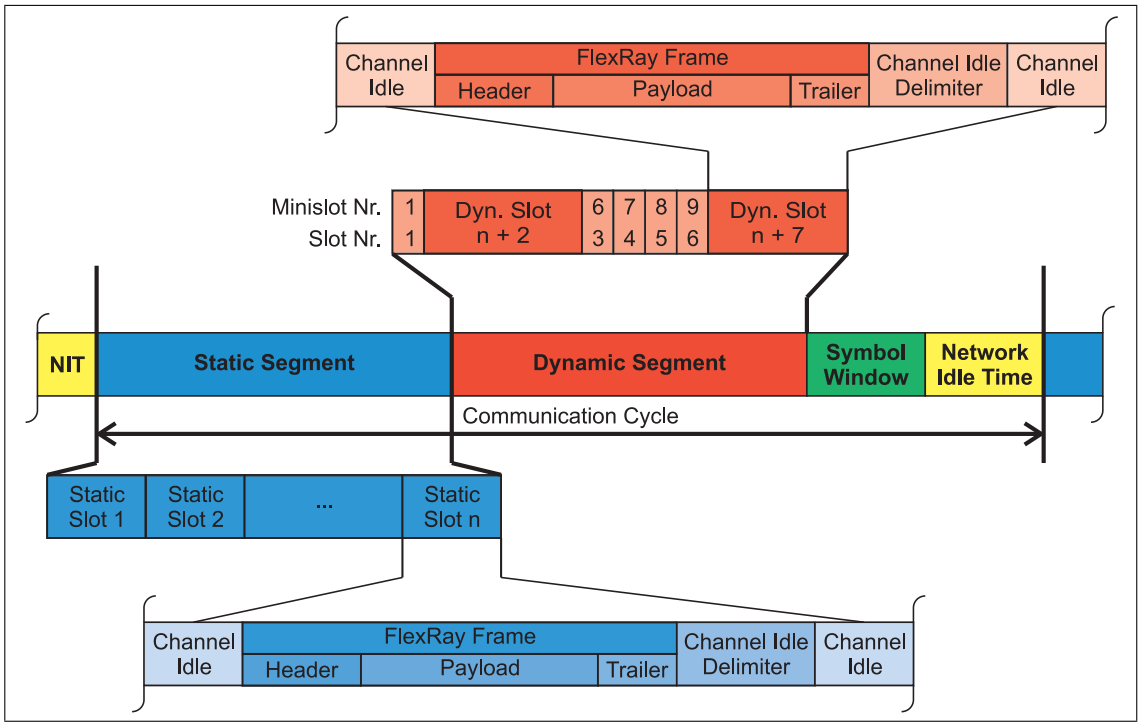


Abbildung 2.1.9: FlexRay Kommunikationszyklus (vgl. FlexRay Consortium, 2005b)

Statisches Segment Die Zugriffsteuerung innerhalb des Statischen Segments erfolgt nach einem TDMA-Verfahren (Time Division Multiple Access). Den einzelnen Knoten in einem Cluster werden ein oder mehrere Slots (Zeitschlitze) fest zugeordnet, in denen der Knoten das Senderecht besitzt. Alle Slots innerhalb dieses Segmentes haben die gleiche Größe. Nachrichten werden als Frames bezeichnet und haben die gleiche Größe. Die Frames werden durch einen Identifier gekennzeichnet (FrameID - aufsteigende Nummerierung beginnend mit 1). Die Slots beginnen auf beiden Kanälen synchron allerdings kann sich die Zuordnung zu den Knoten unterscheiden.

Dynamisches Segment Die Zugriffssteuerung innerhalb des Dynamischen Segments erfolgt nach einem flexiblen Zeitmultiplexverfahren (Flexible Time Division Multiple Access). Dieses wird als Minislot-Verfahren bezeichnet. Dabei wird das Segment in Minislots der gleichen Größe aufgeteilt. Ein Kommunikationsslot im Dynamischen Segment entspricht einem oder mehreren Minislots. Die Frames werden analog zum Statischen Segment über einen Identifier zugeordnet. Die Nummerierung der Slots beginnt mit n+1, wobei n die Anzahl der Slots im Statischen Segment ist.

Das Senden im Dynamischen Segment erfolgt nur nach Bedarf. Ist ein Slot ungenutzt, so hat dieser die Minislot-Größe 1. Wird ein Slot verwendet, so wird dieser

solange um Minislots verlängert, bis das Senden abgeschlossen ist. Hierdurch ergibt sich eine Priorisierung.

Kommunikationsstartup Der Startup eines Kommunikationsclusters, dargestellt in Abbildung 2.1.10, ist die komplizierteste Betriebsphase, da sich hier die Knoten zunächst auf eine gemeinsame Zeitbasis verständigen müssen. Aufgrund der Anforderung Fehlertoleranz gibt es keinen Master.

Es werden zwei Typen von Knoten unterschieden: Coldstart-Knoten, welche einen Cluster-Startup initiieren können und Nicht-Coldstart-Knoten, welche warten bis eine Kommunikation hergestellt wurde. Bei den Coldstart-Knoten werden zwei Rollen unterschieden: Leading (LC) und Following Coldstart (FC) Knoten.

Ablauf: Startup Ein Coldstart-Knoten wartet zwei Zyklen, um zu prüfen ob bereits eine Kommunikation besteht. Falls der Knoten keine Kommunikation registriert, so übernimmt der Knoten die Rolle des Leading-Coldstart-Knoten (LC). Der Knoten startet seine lokale Uhr und somit den Kommunikationszyklus.

Der LC beginnt mit dem Senden in dem ihm zugeordneten Slot. Diese Nachrichten werden von FCs empfangen. Die FCs können anhand der Frame-ID und der empfangenen Nachricht ihre eigene Uhr mit der entsprechenden Slot- und Zyklusnummer initialisieren. Anschließend stellen die FCs sicher, dass ihre Uhr korrekt gestartet wurde und nehmen an der Kommunikation teil. Der LC beendet die Startup-Phase, wenn er Frames von einem FC erhält

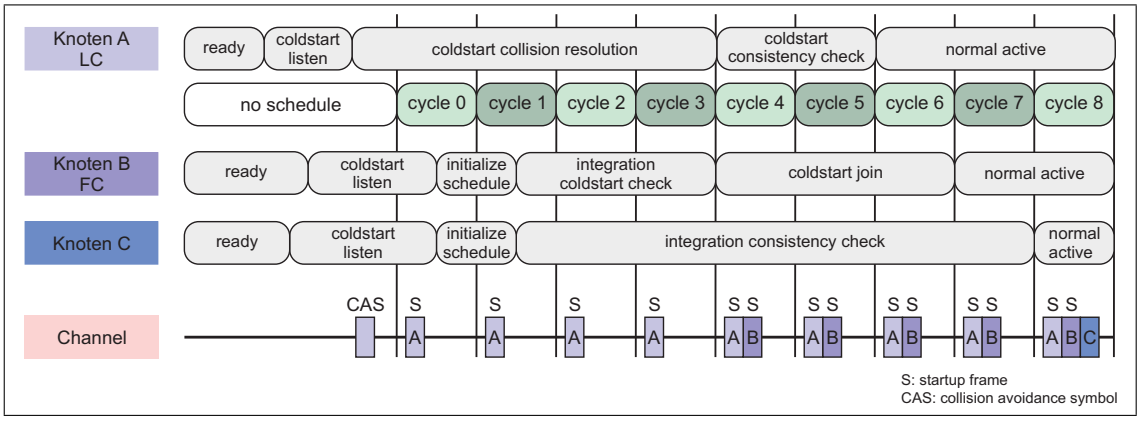


Abbildung 2.1.10: Darstellung des Startup eines FlexRay-Clusters (vgl. FlexRay Consortium, 2005b)

Frame-Format Das FlexRay-Datenframe (Abbildung 2.1.11) gliedert sich in drei Segmente: Header, Payload und Trailer. Innerhalb des Header-Segment werden Steuerinformationen wie beispielsweise Frame-Identifier und Payload-Length übertragen. Das Payload-Segment beinhaltet die Daten und das Trailer-Segment beinhaltet eine CRC-Prüfsumme. Eine ausführlichere Aufschlüsselung der einzelnen Bereiche findet sich im Anhang (A.1).

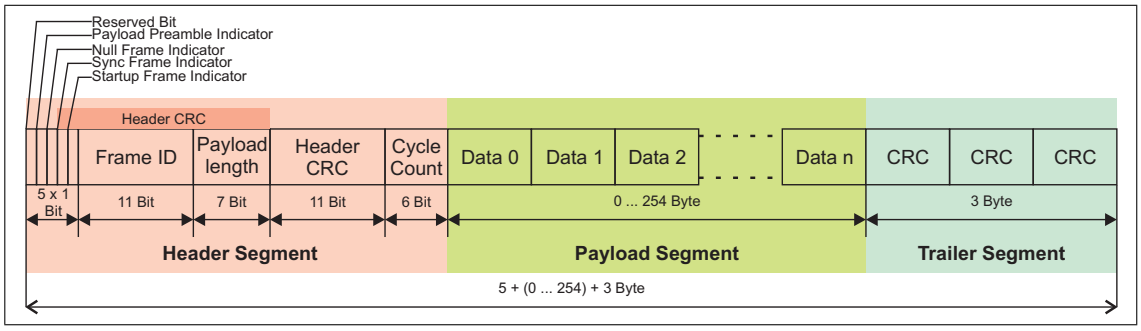


Abbildung 2.1.11: FlexRay-Datenframe (vgl. FlexRay Consortium, 2005b)

2.1.4.5 TTP/C - Time-Triggered Protocol Class C

Das Time Triggered Protocol (TTP) ist ein zeitgesteuertes Kommunikationsprotokoll für verteilte und fehlertolerante Echtzeitsysteme. TTP/C wurde 1993 veröffentlicht, dieses Protokoll mit einem Uhrensynchronisations- und einem Membership-Service erfüllt hohe Anforderungen in den Bereichen Sicherheit, Verfügbarkeit, Zuverlässigkeit sowie Fehlertoleranz und bietet hohe Datenübertragungsraten. (Kopetz & Bauer, 2003)

Der Ursprung des TTP liegt an der Universität Berlin, hier ging es 1979 aus einem Projekt hervor (MARS - Maintable Architecture for Real Time Systems). Die Weiterentwicklung erfolgte im Wesentlichen durch die Firma TTTech[3].

Zwischen TTP/C und FlexRay lassen sich viele Parallelen finden, so erfolgt auch beim TTP/C der Zugriff auf das Busmedium auf Basis eines TDMA-Verfahrens mit einer globalen Zeitbasis. Während sich FlexRay gegen TTP/C im Segment Automotive durchgesetzt hat, kommt TTP/C insbesondere im Bereich Avionik zum Einsatz.

Das zweikanalige Bussystem ermöglicht eine Übertragungsrate von bis zu 25Mbit/s. Im Anhang A.2.2 findet sich eine kurze Beschreibung welche auf (Kopetz, 2001; Obermaisser, 2012; Kopetz & Bauer, 2003) und der Spezifikation (ttaGroup, 2002) beruht.

2.1.4.6 Vergleichende Betrachtung ausgewählter Bussysteme

Bei der Gegenüberstellung und dem Vergleich der vorgestellten Bussysteme wird auf Eigenschaften Wert gelegt, die im Rahmen der Modellierung der Kommunikation, der Modellierung von Kommunikationssystemen und der Analyse solcher Systeme von Bedeutung sind. Ein ähnlicher Vergleich findet sich auch in der Arbeit von Jentzsch (2003).

2.1.4.6.1 Kriterien zur Gegenüberstellung der Bussysteme Die Kriterien für die Gegenüberstellung der Bussysteme werden in drei Gruppen eingeteilt und basieren im Wesentlichen auf (Gruhler, 2001). Die erste Gruppe „Grundlegende Eigenschaften“ beinhaltet allgemeine Eigenschaften die eingeschränkte Auswirkungen auf die Modellierung besitzen. Sehr relevant für die Modellierung ist die zweite Gruppe „Buszugriffs-

[3] https://www.tttech.com/ [08.01.2015]

und Kommunikationsverhalten“, da auf diesen Mechanismen der Schwerpunkt der Modellierung liegt. In der dritten Gruppe finden sich Eigenschaften zu Fehler- und Sicherheitsmechanismen, welche insbesondere bei der Modellierung von Gesamtsystemen, Fehlerinjektion u.ä. von Relevanz sind.

Grundlegende Eigenschaften Zu den allgemeinen Eigenschaften zählen das typische Einsatzgebiet, die Einordnung in der Automatisierungspyramide (siehe 2.1.7) sowie die Topologie des Kommunikationsnetzwerkes. Ergänzt werden diese durch wichtige numerische Parameter wie die maximale Anzahl der Busteilnehmer sowie die Brutto-Datenübertragungsrate. Abschließend wird noch das verwendete Übertragungsmedium betrachtet.

Typisches Einsatzgebiet Anwendungsbereich sowie Ebene der Automatisierungspyramide in denen das Kommunikationssystem einsetzbar ist.

Topologie Strukturen des Kommunikationsnetzes.

Anzahl Busteilnehmer Maximale Anzahl der möglichen Kommunikationsteilnehmer in einem Netzwerk.

Datenübertragungsrate Maximale Bruttodatenübertragungsrate des Bussystems.

Übertragungsmedium Vom Bussystem verwendbare Busmedien.

Buszugriff und Kommunikationsverhalten Zu den Eigenschaften im Bereich Buszugriff und Kommunikationsverhalten gehören neben dem Verfahren zum Buszugriff, die Rollen der Teilnehmer bei der Kommunikation sowie deren Synchronisation, die Art der Kommunikation sowie Adressierung, Priorisierung und nachrichtenbezogene Eigenschaften. Mitbetrachtet wird auch die Möglichkeit des Hinzufügens von neuen Teilnehmern.

Kommunikationssteuerung Hierarchische oder gleichberechtigte Teilnehmer.

Buszugriff Verfahren zur Regelung des Zugriffs auf das Übertragungsmedium.

Ereignis- oder zustandsorientierte Kommunikation Die Kommunikation erfolgt auf Basis einer Ereignissteuerung oder Zeitsteuerung (Zustandssteuerung).

Adressierungsarten Art der Adressierung von Telegrammen und Teilnehmern.

Prioritäten Vorhandensein von Prioritäten bei der Vergabe der Sendeberechtigung (die Wichtung erfolgt teilnehmer- oder nachrichtenbezogen).

Nachrichtenlängen Zur Versendung von Nachrichten (Messages) verwendete Anzahl von Bits oder Bytes.

Synchronisation der Zeitbasen Das angewendete Verfahren zur Synchronisation der Busteilnehmer.

Nachrichtenbestätigung Signalisierung einer erfolgreichen bzw. fehlgeschlagenen Datenübertragung.

Fehler und Sicherheitsmechanismen Die letzte Gruppe beinhaltet Eigenschaften die sich aus den Anwendungsbereichen ableiten lassen sowie Einfluss auf die Gesamtsystemmodellierung und Simulation von konkreten Systemszenarien besitzen. Dies sind insbesondere fehlerbezogene Eigenschaften wie Fehlererkennung, Verhalten bei allgemeinen Fehlern und Teilnehmerausfall sowie Sicherheitsaspekte wie Redundanz und Schutz vor Busmonopolisierung.

Fehlererkennung Verfahren zur Erkennung von fehlerhaften Telegrammen (Paritätsbit, CRC etc.).

Verhalten bei Fehlern Verhalten der Teilnehmer bei Übertragungsfehlern (Anzahl der Sendewiederholungen, Fehlersignalisierung u.ä.).

Verhalten bei Teilnehmerausfall Verhalten des Systems und der Teilnehmer bei Ausfall eines oder mehrerer Teilnehmer.

Redundanz Verfügbarkeit von redundanten Übertragungskanälen und Verwaltung redundanter Teilnehmer.

Schutz vor Busmonopolisierung Mechanismus zum Schutz vor einer dauerhaften Belegung des Busses durch einen unkontrolliert sendenden (fehlerhaften) Teilnehmer.

2.1.4.6.2 Gegenüberstellung der Bussysteme Bei der Gegenüberstellung der Bussysteme werden: LIN (Grzemba & Wense, 2005; LIN Consortium, 2010; Zimmermann & Schmidgall, 2011), CAN (Robert Bosch GmbH, 1991; Etschberger, 2002; Zimmermann & Schmidgall, 2011), FlexRay (FlexRay Consortium, 2005b, 2005a; Rausch, 2008; Zimmermann & Schmidgall, 2011) und TTP/C (Obermaisser, 2012; ttaGroup, 2002) berücksichtigt. Die ersten drei haben dabei das typische Einsatzgebiet Fahrzeugtechnik gemeinsam, wobei insbesondere der CAN zusätzlich in weiteren Gebieten wie Gebäudeautomatisierung, Automatisierungstechnik, Flugzeugtechnik usw. zum Einsatz kommt. Ebenso wie FlexRay ist der TTP/C für sicherheitskritische Systeme gedacht, findet aber hauptsächlich in der Luft- und Raumfahrttechnik sowie im Bereich Schienenverkehr und Industrieautomation Anwendung. Die beiden komplexen Kommunikationssysteme FlexRay und TTP/C erlauben im Gegensatz zu den beiden anderen neben Bus auch Stern, Baum oder kombinierte Topologien. In diesen so gestalteten Netzen findet sich bei LIN, FlexRay und TTP/C eine Beschränkung auf bis zu 64 Teilnehmer. Die Leistungsfähigkeit bei der Datenübertragung differiert zwischen den Systemen stark, so bietet die Ein-Draht-Leitung des LIN lediglich eine Übertragungsrate von 1 bis 20 kBit/s. Die nächste Stufe bildet der CAN mit einer Rate von bis zu 1 MBit/s und einer verdrillten Zwei-Draht-Leitung. Der zwei-kanalige Aufbau des FlexRay erlaubt bis zu 10 MBit/s pro Kanal und kann ebenso wie TTP/C eine verdrillte Zwei-Draht-Leitung oder aber auch einen Lichtwellenleiter als Übertragungsmedium verwenden. Im Hinblick auf die Übertragungsrate ist der TTP/C mit 25 MBit/s von diesen vieren am leistungsstärksten.

Im Bereich der Zugriffssteuerung handelt es sich lediglich beim LIN um ein Master-Slave-System, während die anderen Multi-Master-Systeme sind. LIN und CAN arbeiten rein ereignisgesteuert und verwenden beim Buszugriff ein Polling-Verfahren

(LIN) bzw. ein CSMA/CA-Verfahren (CAN). Der TTP/C nutzt für seine reine Zeitsteuerung ein TDMA-Verfahren. Eine Sonderstellung nimmt in diesem Vergleich der FlexRay ein, der sowohl ein zeit- (TDMA) als auch ereignisgesteuertes (FTDMA) Zugriffsverfahren realisiert.

In den zeitgesteuerten Systemen FlexRay und TTP/C erfolgt die Adressierung über die Zuordnung von Kommunikationsslots zu Teilnehmern, wobei lediglich innerhalb des FlexRay im Dynamischen Segment, welches den ereignisgesteuerten Protokollteil realisiert, durch die Slot-ID eine Priorisierung erfolgt.

Für den zeitgesteuerten Buszugriff sind synchrone Zeitbasen von enormer Wichtigkeit, FlexRay nutzt einen Midpoint-Algorithmus und TTP/C einen Average-Algorithmus für die Synchronisation der Uhren der Teilnehmer. Für CAN und LIN ist eine Synchronisation der Bitzeiten ausreichend. Innerhalb des CAN-Bus bildet die nachrichtenbasierte Adressierung in Verbindung mit der darauf bezogenen Priorisierung die Grundlage für die Realisierung des Buszugriffs.

Nachrichten beim LIN und CAN können bis zu 8 Byte Nutzdaten enthalten und im Falle des CAN-Bus erfolgt eine stationsneutrale Empfangsbestätigung. Eine Art Empfangsbestätigung ist auch im TTP/C vorhanden, die übertragenen Nutzdaten sind mit 240 Byte um sechs Byte geringer als beim FlexRay.

Die Sicherheitsmechanismen und das Verhalten bei Fehlern sind bei den vier Protokollen sehr unterschiedlich. So sichern die LIN-Slaves Diagnoseinformationen die vom Master abgerufen werden können. Dabei wird der Nachrichten-Header durch zwei Paritätsbits und die Daten über eine Prüfsumme abgesichert. Der CAN kennzeichnet fehlerhafte Nachrichten durch ein Flag, hierdurch werden Fehler dem gesamten System bekannt gemacht und die Nachricht zerstört. Die Absicherung erfolgt dabei durch eine CRC. Zusätzlich prüft auch der Sender jeweils den Bus-Pegel. Auch TTP/C und FlexRay nutzen eine CRC zur Fehlerprüfung, die erkannten Fehler werden der Applikationsebene gemeldet, darauf folgende weitere Verarbeitungsschritte bleiben der Anwendungsebene vorbehalten. Gibt es einen Komplettausfall eines Teilnehmers, so arbeiten alle Busse weiter. Der Anwendungsbereich sicherheitskritische Systeme spiegelt sich bei FlexRay und TTP/C zum Beispiel durch Redundanz der Kanäle und dem Schutz vor einer Busmonopolisierung wieder.

Eine kurze Übersicht der Gegenüberstellung findet sich in Tabelle 2.2.

Tabelle 2.2: Gegenüberstellung ausgewählter Bussysteme (tabellarisch)

Kriterien	**LIN**	**CAN**	**FlexRay**	**TTP/C**
Typisches Einsatzgebiet	primär in der Fahrzeugtechnik als Subbus des CAN, Industrieautomation	Fahrzeugtechnik, medizinische Geräte, Flugzeuge, Automatisierungstechnik, Werkzeugmaschinen, Textilmaschinen, Gebäudeautomatisierung	flexible Verkabelung sicherheitskritischer Systeme in Kraftfahrzeugen, Einsatz in anderen Bereichen möglich	Vernetzung sicherheitskritischer Systemelektronik in der Luft- und Raumfahrt, im Schienenverkehr und in der Industrieautomation
Topologie	Bus	Bus (mit Repeater auch Baum)	Bus, Stern, Kombinationen	Bus, Stern, Kombinationen
Maximale Anzahl Busteilnehmer	1 Master, theoretisch 63 Slaves, max. 15 Slaves empfohlen	keine Begrenzung	64	64
Datenübertragungsrate	1 kBit/s bis 20 kBit/s	50 kBit/s,100 kBit/s, 250 kBit/s, kBit/s, 1 MBit/s	bis 10 MBit/s (pro Kanal)	bis 25 MBit/s
Übertragungsmedium	Eindrahtleitung	Twisted Pair	Twisted Pair, LWL	Twisted Pair, LWL
Kommunikationssteuerung	Master-Slave	Multi-Master	Multi-Master	Multi-Master
Buszugriff	Polling	CSMA/CA	TDMA, FTDMA	TDMA
Ereignis- oder zustandsorientierte Kommunikation	ereignisgesteuertes System	ereignisgesteuertes System	ereignis- und zeitgesteuertes System	zeitgesteuertes System
Adressierungsarten	indirekt durch 6 Bit Identifier	11 Bit Message-Identifier (CAN 2.0a), 29 Bit Message-Identifier (CAN 2.0b)	Broadcast-Protokoll, Teilnehmer wird Slot zugeordnet	Broadcast-Protokoll, Teilnehmer wird Slot-Positionen zugeordnet
Prioritäten	Keine	Priorität wird durch den Nachrichten-Identifier bestimmt	Slot-ID bestimmt Priorität (dyn. Segment)	Keine
Nachrichtenlängen	Nutzdaten: 0-8 Byte; Steuerinformationen: 3 Byte; Synch Break: min. 14 Bit; jedes Byte wird UART-codiert (mit 10 Bit) übertragen	Datenframe: 0-8 Byte; Steuerinformationen: 47-67 Bit; variable Anzahl Stuffbits	Nutzdaten: 0-254 Byte, Steuerinformationen: 8 Byte	Nutzdaten: 0-240 Byte, Steuerinformationen: 28-156 Bit
Synchronisation der Zeitbasen	jede Master-Slave Runde beginnt mit einem Synch Break gefolgt von einem 1 Byte Synch Field	Synchronisation der Bitzeiten über Startbit	Uhrenkorrektur durch Fault-Tolerant Midpoint Algorithmus	Uhrenkorrektur durch Fault-Tolerant Average Algorithmus
Nachrichtenbestätigung	vom Protokoll nicht unterstützt	Sender erhält „stationsneutrale positive Empfangsbestätigung“ in einem 2 Bit Bestätigungsfeld, lokal gestörte Teilnehmer senden ein Fehlertelegramm	Information nicht verfügbar	Implizit durch die Membership Informationen

Fehler-erkennung	2 Paritätsbits schützen Identifier; Prüfsumme über alle Datenbytes	Cyclic Redundancy Check (CRC), Format Check, Sender überprüft eigene Übertragung	3 Byte CRC, 9 Bit Header-CRC, 7 Bit Längenkontrolle, 6 Bit Cycle Counter	Cyclic Redundancy Check (CRC), Überprüfung des C-State
Verhalten bei Fehlern	Erkannte Fehler werden von den Slaves als Diagnoseinformationen gespeichert, die vom Master abgerufen werden können.	Eine als fehlerhaft erkannte Nachricht wird mit einem Fehlerflag gekennzeichnet und dadurch für alle Netzwerkstationen ungültig (Error-Frame)	Fehler werden dem Host gemeldet	Fehler werden dem Host gemeldet.
Verhalten bei Teilnehmer-ausfall	Der Bus arbeitet weiter. Der Master überwacht die Kommunikation der Slaves und meldet Fehler der Anwendungsebene	Der Bus arbeitet weiter, wenn eine Station in den „bus off“ Modus schaltet.	Bus arbeitet weiter, Fehler müssen erkannt werden und Diagnose an Host übermittelt werden; Applikationsschicht (bzw. FT-COM Schicht) ist für das Management von Redundanz verantwortlich	Bus arbeitet weiter, Applikationsebene und alle anderen Teilnehmer werden informiert
Redundanz	nicht unterstützt	nicht unterstützt	Zweiter Übertragungskanal optional	2 Übertragungskanäle
Schutz vor Busmonopolisierung	nicht unterstützt	nicht unterstützt	Je Kanal ein unabhängiger Bus Guardian im Netzknoten	Unabhängiger Bus Guardian (lokal oder zentral in einem Sternknoten)

2.1.5 Beschreibung und Spezifikation der Netzwerke in einem Automobil

Bei der Entwicklung von komplexen vernetzten Systemen, wie im Automobilbereich üblich, ist eine Vielzahl von unterschiedlichen Herstellern und Entwicklergruppen beteiligt. Die Kommunikationsarchitektur ist eine zentrale Schnittstelle dieser Entwicklergruppen und muss einheitlich beschrieben werden. Für die einzelnen Bussysteme werden jeweils spezielle Beschreibungsmittel eingesetzt. Diese beinhalten die Spezifikation der Kommunikationsarchitektur und die Definition von Nachrichten, Teilnehmern und weiteren Parametern.

Im Falle des CAN-Bus hat sich das dbc-Format durchgesetzt (Vector Informatik GmbH, 2014; Zimmermann & Schmidgall, 2011). Der LIN-Bus berücksichtigt eine Beschreibung bereits innerhalb seiner Spezifikation, die LIN Description Files (LDF) (Grzemba & Wense, 2005). Eine Sonderstellung nimmt der FlexRay ein, zwar wurde hier auch eine spezifische Beschreibung nämlich FIBEX (Fieldbus Exchange Format) (Zimmermann & Schmidgall, 2011) entwickelt, diese ist aber ausgelegt für die Beschreibung der Kommunikation im gesamten Automobil und unterstützt auch andere Bussysteme (LIN, CAN, MOST, FlexRay). So kann mit einer FIBEX-Datei die gesamte Kommunikation in einem Fahrzeug beschrieben werden. Nachfolgend erfolgt eine kurze Betrachtung der Beschreibungen für CAN und für FlexRay, da diese Bussysteme im Folgenden als Referenzbeispiel angeführt werden.

2.1.5.1 CAN database (dbc) / CAN-Matrix

CAN database Dateien enthalten Informationen über Aufbau, Struktur und Inhalt von CAN-Netzwerken. Die Beschreibung von Botschaften, Signalen, Steuergeräten, Netzknoten und Netzwerken erfolgt dabei auf textueller Basis.

Das Datenformat *dbc* wurde durch die Vector Informatik GmbH entwickelt. Für den Austausch von Beschreibungen von Van-Netzwerken hat es sich insbesondere im Automobilbereich zum Quasi-Standard entwickelt.

Für jedes einzelne Signal sind Informationen zur Konvertierung in technische Einheiten notwendig, dazu zählen: Kanal-Name, Position und Größe des Kanals in einer Nachricht, Byte-Order, Datentyp, Skalierung und Einheit, Wertebereich, Standard-Wert und Kommentar. Spezifisch gilt dies ausschließlich für CAN-Netzwerke.

2.1.5.2 Fieldbus Exchange Format - FIBEX

FIBEX (Fieldbus Exchange Format) ist ein durch die ASAM standardisiertes XML-Schema zur Beschreibung von kompletten Fahrzeugnetzwerken (ASAM, 2007; Zimmermann & Schmidgall, 2011). Allgemein kann es zur Beschreibung von Kommunikationsnetzen im gesamten Feldbus-Segment genutzt werden und hat sich als Quasi-Standard für die Beschreibung von FlexRay-Clustern etabliert. Für die einzelnen Bus-Systeme gibt es zur Beschreibung der unterschiedlichen Konfigurationen jeweils spezielle protokollbezogene Erweiterungen.

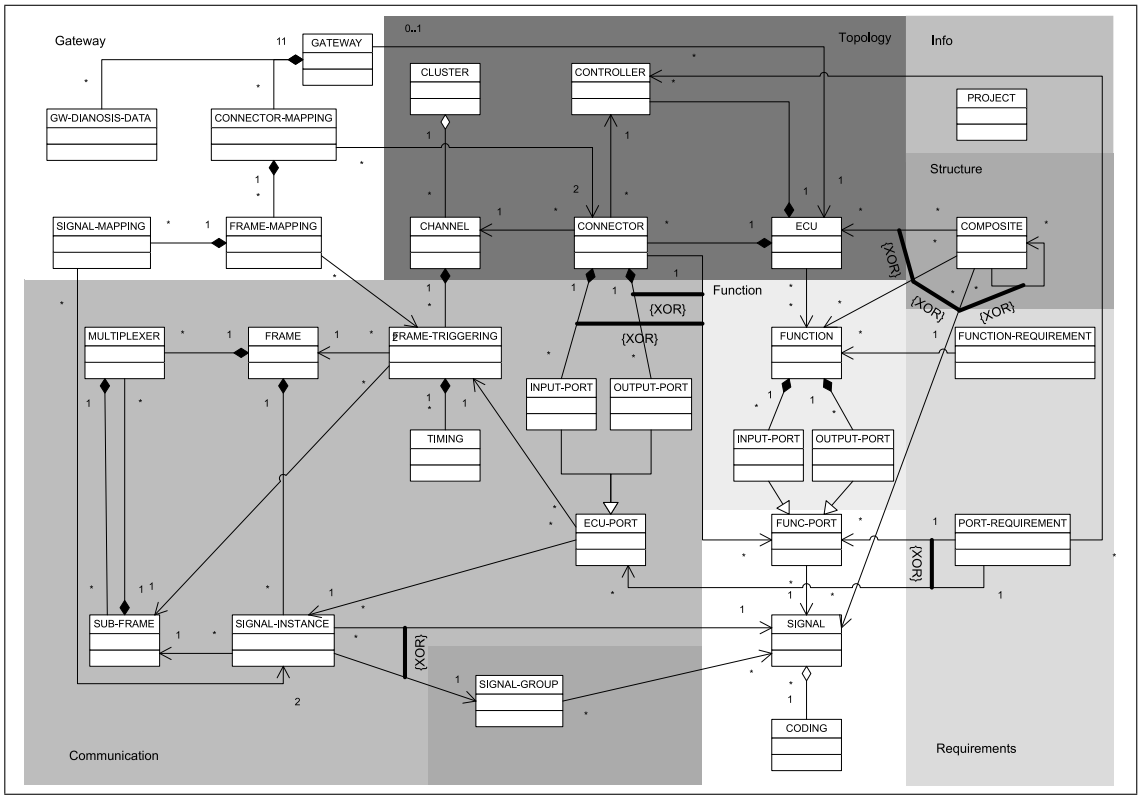


Abbildung 2.1.12: Aufbau des FIBEX XML-Schema (ASAM, 2007)

Durch FIBEX wird eine detaillierte Beschreibung der Vernetzungstopologie von Fahrzeugen sowie der Kommunikation innerhalb und zwischen den Netzwerken ermöglicht. Das Hauptziel ist die Bereitstellung eines einfachen und klar definierten Datenaustauschformates.

Die Struktur des Standards ist in dem UML-Diagramm in Abbildung 2.1.12 dargestellt. Aus der Grafik ist der Aufbau der Hauptelemente zu entnehmen. Die Beschreibung kann zunächst in Topologie, Funktionen, Kommunikation und Gateway unterteilt werden. In der Systemtopologie wird die Hardware Architektur des Systems durch die Elemente Cluster, Channel (Bus) mit Frames, ECU mit Signalen und Funktionen, Communication Controller und Gateway beschrieben.

Die Kommunikation wird auf Frameebene festgelegt, dabei werden ausgetauschte Frames samt Timings erfasst. Den Frames werden jeweils Signale sowie die Sender- und Empfängerrelationen zugeordnet. Diese sind abhängig vom konkreten Kommunikationsprotokoll.

Im FIBEX-Standard (ASAM, 2007) werden die Elemente Cluster, Channel, Connector, Controller, ECU und Gateway folgendermaßen definiert:

CLUSTER Ein Cluster beschreibt eine Gruppe von Steuergeräten, die durch ein Kommunikationsmedium verbunden sind und eine beliebige Topologie (Bus, Stern, Ring , ...) bilden. Die Knoten in dem Cluster verwenden das gleiche Kommunikationsprotokoll, dieses kann ereignisgesteuert, zeitgesteuert oder eine Kombination von beiden sein.

CHANNEL Das Kommunikationsmedium eines Clusters besteht aus einem oder mehreren (redundanten) Kommunikationskanälen (communication channel). Der Kanal (Channel) verknüpft die Steuergeräte eines Clusters physikalisch: Eine ECU ist Teil eines Clusters, wenn mindestens ein Controller existiert, der mit mindestens einem Kanal des Clusters verbunden ist.

CONNECTOR Ein Connector ist ein symbolisches Element, diese werden verwendet um die Bus-Schnittstellen der Steuergeräte zu beschreiben und das Sende- und Empfangsverhalten zu spezifizieren. Mit einem Connector kann eine ECU einem Channel zugeordnet werden, bevor der Controller definiert wurde.

CONTROLLER Der Controller (Communication-Controller) ist eine spezielle Hardware mit deren Hilfe das Senden und Empfangen von Frames über das Kommunikationsmedium erfolgt.

ECU Die ECU (Electronic Control Unit - Steuergerät) umfasst alles was eine reale ECU ausmacht, abgesehen vom Communication Controller. Die ECU enthält eine oder mehrere Host-Prozessoren welche einen Teil der verteilten Anwendung ausführen. Eine ECU kann als Gateway arbeiten, wenn es mit zwei oder mehreren unterschiedlichen Clustern mit zwei oder mehreren Communication Controllern verbunden ist.

GATEWAY Ein Gateway ist ein Steuergerät welches mit zwei oder mehreren Clustern verbunden ist und Frames oder Signale zwischen diesen abbildet.

2.1.6 Verbindung von Bussystemen im Automobil - Gateway

Im Automotive Bereich kommen aufgrund unterschiedlicher Anforderungen, Eigenschaften und Kosten unterschiedliche Kommunikationsprotokolle zum Einsatz. Einzelne Anwendungen und Funktionen erfordern den Austausch von Daten über die Grenzen einzelner Kommunikationsnetze (Busse, Kommunikationsprotokolle) hinaus. Die Verbindung der unterschiedlichen Bussysteme erfolgt dabei über Gateways. Ein einzelnes zentrales Gateway oder mehrere Gateways werden zum Datenaustausch zwischen den einzelnen Kommunikationsnetzen CAN, LIN, MOST und FlexRay verwendet. Aus Sicht eines einzelnen Netzwerkes erscheint ein Gateway als ein einzelnes angeschlossenes Steuergerät

Gateways sind spezielle Netzwerkknoten, welche die Hauptaufgabe besitzen zwischen unterschiedlichen Netzen zu vermitteln. Sie bilden dabei den Übergangspunkt zwischen zwei oder mehreren Netzwerken, welche unterschiedliche Protokolle, Datenformate, Bitrepräsentationen o.ä. verwenden. Ein Gateway-Element umfasst daher beispielsweise einen Protokollumsetzer, eine Flusskontrolle, eine Routing-Einheit und einen Signalumsetzer. Einem Gateway werden die Schichten 5 bis 7 des ISO/OSI-Referenzmodells zugeordnet. (Reif, 2006; Lorenz, 2008)

2.1.6.1 Basis Struktur

Ein Gateway ist applikationsspezifisch und somit abhängig vom Einsatzgebiet. Für Anforderungen an Gateways kann man sich innerhalb der Automotive Domäne an AUTOSAR (Automotive Open System Architecture) orientieren (AUTOSAR GbR, 2008a, 2008b). Für weiterführende Informationen zu AUTOSAR sei an dieser Stelle auf (Zimmermann & Schmidgall, 2011; Kindel & Friedrich, 2009) sowie die Internetpräsenz der AUTOSAR-Organisation verwiesen[4].

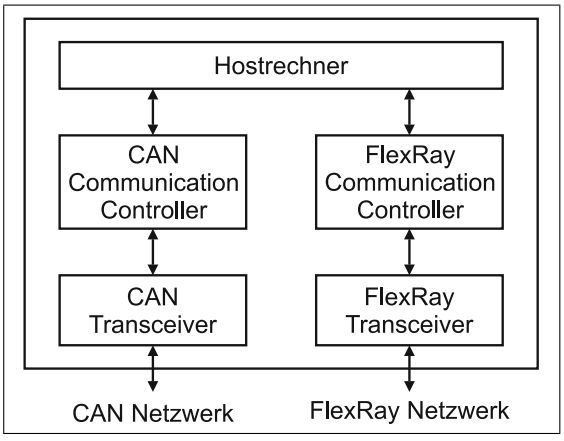


Abbildung 2.1.13: Beispiel für eine Gateway Struktur mit FlexRay und CAN

Ein Gateway ist ein spezielles Steuergerät mit mehreren Kommunikationsschnittstellen (vgl. Seo, Lee, Hwang & Jeon, 2006). In Abbildung 2.1.13 wird als Beispiel

[4] `http://www.autosar.org/` [08.01.2015]

ein CAN-FlexRay-Gateway dargestellt. Das Gerät umfasst einen Host-Controller, zwei Protokoll-Communication-Controller (CAN und FlexRay) und zwei Protokoll-Transceiver (CAN und FlexRay). Hauptaufgabe des Host-Controllers sind Adress- und Protokollumsetzung. Diese Routinginformationen sowie Signalinformationen sind im Host hinterlegt.

Ein Gateway muss unter anderem folgende Funktionalitäten unterstützen: Adressierung, Nachrichtenfilterung, Informationsflusssteuerung und Nachrichtenpufferung. Zwischen zwei Netzen erfolgt die direkte Kommunikation durch eine netzübergreifende Adressierung. Bei der Weiterleitung erlauben statische und dynamische Nachrichtenfilter eine Blockierung von für das Zielnetzwerk nicht relevanten Daten. Die Filterung kann innerhalb des Gateways auch auf Anwendungsebene erfolgen. Im Rahmen der Steuerung des Informationsflusses kann eine Nachrichtenumwandlung erfolgen. Beispielsweise eine CAN-Ereignisnachricht wird in eine periodische FlexRay-Statusnachricht gewandelt, oder auch eine Prioritätsanpassung, wenn beispielsweise zwischen zwei CAN-Netzen übertragen wird. Durch eine lokale Pufferung der Nachrichten können Überlast-Situationen im Zielnetzwerk vermieden werden.

Die Nachrichtenweiterleitung kann auf unterschiedlichen Ebenen erfolgen. Diese unterschiedlichen Aufgaben werden nachfolgend unter Zuhilfenahme von AUTOSAR erläutert.

2.1.6.2 AUTOSAR Gateway

AUTOSAR hat das Ziel durch eine einheitliche Software-Architektur mit standardisierten Schnittstellen die Komplexität von Software und elektronischen Systemen im Automobilbereich beherrschbar zu gestalten.(Zimmermann & Schmidgall, 2011) Durch diese Standardisierung ist gleichzeitig eine Kostensenkung sowie eine Qualitätsverbesserung möglich. Nachfolgend werden drei Funktionstypen von Gateways kurz erläutert. (Hörner, 2007)

2.1.6.2.1 PDU-Gateway Das PDU-Gateway (Protocol Data Unit) ist innerhalb der AUTOSAR-Architektur Teil des PDU-Routers und leitet gesamte Datenpakete beim Empfang direkt zu einem anderen Netzwerk weiter. Dadurch ergeben sich Einschränkungen, so müssen die PDUs in Ziel- und Quell-Netzwerk identisch definiert sein (z.B. Länge und Inhalt). Durch die direkte Weiterleitung können Sendeart und Zykluszeit nicht angepasst werden.

2.1.6.2.2 Signal-Gateway Auf Ebene des Signal-Gateway werden einzelne Signale zwischen den Netzwerken vermittelt. Eine empfangene PDU wird dabei in Signale zerlegt und für das Zielnetzwerk neu zusammengesetzt. Hierdurch ist neben der Anpassung der Signalzusammensetzung und -position innerhalb einer Nachricht auch eine Veränderung der Sendeart und Zykluszeit möglich.

2.1.6.2.3 TP-Gateway Das TP-Gateway (Transport Protokoll) dient zur automatischen Weiterleitung von Diagnosedaten zwischen Netzwerken und ist ebenso Bestandteil des PDU-Routers. Hierbei erfolgt der Datenaustausch auf der Ebene der

Transportprotokolle. Bei CAN und FlexRay ist das Transport-Protokoll anders als bei LIN nicht Teil des Kommunikationsprotokolls sondern ein zusätzlicher Baustein, der in der AUTOSAR-Architektur auch berücksichtigt wird. Zur Reduzierung des lokalen Speicherbedarfs werden Daten weitergeleitet, bevor das gesamte Datenpaket empfangen wurde.

2.1.6.3 Beispiele für ein Gateway

Auf dem Markt gibt es mittlerweile eine Vielzahl von Gateway-Komponenten. Insbesondere für die Entwicklung und das Prototyping finden sich vorgefertigte Module von verschiedenen Anbietern. Mit der Beschränkung auf das Segment Automotive seien an dieser Stelle drei willkürliche Beispiele genannt.

CAN-GW100RS232 von IXXAT Das *CAN-GW100 RS232*[5] schließt Geräte mit einer seriellen RS232-Schnittstelle an den CAN-Bus an. Die empfangenen CAN-Daten werden auf die RS232 Schnittstelle übertragen. Die über die RS232-Schnittstelle gesendeten Daten werden in konfigurierbare CAN-Telegramme verpackt.

CW-121 - CANnect II von CANway Technology GmbH Das CANnect II[6] ist ein CAN-Gateway und realisiert eine Verbindung von unterschiedlichen CAN-Bussen. Das Gateway arbeitet als normaler Knoten, welcher Botschaften empfangen, verarbeiten und senden kann.

AT-FlexRay-Gateway 5003 von Silver Atena Das *AT-FlexRay-Gateway 5003*[7] ist ein konfigurierbares Gateway-Modul und unterstützt die Bussysteme FlexRay, CAN, LIN und Ethernet.

Der Entwurf von leistungsfähigen Gateways insbesondere im Zusammenhang mit zeitgesteuerter Kommunikation und den beiden Kommunikationsprotokollen FlexRay und CAN ist eine relevante Fragestellung zu der sich in wissenschaftlichen Arbeiten unterschiedliche Ansätze finden. Zur Betrachtung von Systemeigenschaften muss die Art und Weise der Vermittlung zwischen einzelnen Netzwerken betrachtet werden.

Shaheen, Heffernan und Leen (2007) befasst sich allgemein mit der Architektur eines Gateway für zeitgesteuerte Netze (u.a. FlexRay, TTP/C, TTCAN) und der Echtzeitkommunikation zwischen unterschiedlichen Kommunikationsnetzen. Lorenz (2008) entwickelt ein Gateway-Konzept zur Anwendung im Segment Automotive, welches basierend auf einem einheitlichen Beschreibungsformat die Konfiguration sowie die Verifikation eines Gateways ermöglicht bzw. unterstützt. Insbesondere zu spezifischen Gateways zur Kopplung der Bussysteme CAN und FlexRay finden sich verschiedene Arbeiten (Seo, Kim, Hwang, Kwon & Jeon, 2012; Somers, 2009; Seo et al., 2006). Neben diesen Gateway-Architekturen, welche sich mit den klassischen Bussystemen befassen, spielt zunehmend auch die fahrzeugexterne Kommunikation oder

[5] `http://www.ixxat.de/can-gw100_rs232_de.html` [08.01.2015]
[6] `http://www.canway.de/de/produkte/cw-100er-serie-bustechnik/cw-121-cannect-ii` [08.01.2015]
[7] `http://www.silver-atena.de/produkte/simulatoren/flexray-gateway/` [08.01.2015]

auch die Verwendung, Integration und Anbindung weiterer Kommunikationsprotokolle eine Rolle. Als Beispiel sei hier die Verknüpfung der klassischen automotive Bussysteme mit IP/Ethernet-basierter Kommunikation genannt.(Zinner, Noebauer, Gallner, Seitz & Waas, 2011)

Einen wichtigen Baustein bei der Betrachtung von Systemeigenschaften und dem Entwurf von Netzwerken bildet somit das Gateway bzw. allgemein die Kopplung unterschiedlicher Teilnetze zur Realisierung einer oder mehrerer Anwendungen bzw. Aufgaben und Funktionen.

2.1.7 Verknüpfungen zwischen der Automatisierungspyramide und der Domäne Automotive

Ein solcher Verbund von elektrischen (vernetzten) Komponenten, welcher u.a. für Steuerungsaufgaben eingesetzt wird, findet sich nicht nur in Fahrzeugen sondern in einer Vielzahl von Systemen. Die in einem Fahrzeug ablaufende Verarbeitung kann allgemein der Prozessdatenverarbeitung zugeordnet werden bzw. lassen sich zu dieser Parallelen finden. Der im Verlauf dargestellte Modellierungsansatz wird abschließend auf die Domäne Mess- und Automatisierungstechnik erweitert. Diese lässt sich im Allgmeinen ebenso der Prozessdatenverarbeitung zuordnen. Nachfolgend soll exemplarisch eine Verknüpfung der Domäne Automotive sowie der Prozessdatenverarbeitung anhand der Automatisierungspyramide vorgenommen werden.

Die Automatisierungspyramide hat sich in der Prozessdatenverarbeitung zur Einordnung von Techniken und Systemen sowie zur Darstellung der Informationen etabliert. Jeder Ebene werden dabei spezifische Aufgaben zugeordnet. Man findet unterschiedliche Ausprägungen der Automatisierungspyramide abhängig von der konkret betrachteten Systemausprägung. Allgemein finden sich die Ebenen: Unternehmensebene, Betriebsleitebene, Prozessleitebene, Steuerungsebene, Feldebene und Sensor-Aktor-Ebene. Kontextabhängig entfallen häufig insbesondere die oberen und unteren Ebenen, oder werden im Falle von Unternehmens- und Betriebsleitebene zu einer Managementebene kombiniert. In die Feldebene wird ebenso teilweise die Sensor-Aktor-Ebene integriert. Ein Beispiel für eine zusätzlich eingeführte Ebene ist die Zellebene zwischen Steuerungs- und Leitebene.

In (Kagermann, Wahlster & Helbig, 2012) werden die vier Ebenen Planungsebene, Produktionsleitebene, Steuerebene und Feldebene definiert. In (Haußner, Elger & Trautmann, 2010) finden sich die folgenden Ebenen: MES[8]/ERP[9]-Ebene (Betriebsleitebene/Unternehmensleitebene), Prozess-/Leitebene, Steuerungsebene sowie Feldebene und in (Zacher & Reuter, 2011) Betriebsleitebene, Prozessleitebene, Feldebene und der Prozess selbst.

Zudem wird in der DIN EN 62264 zur Integration von Produktions-, Leit- und EDV-Systemen eine Definition von fünf Ebenen vorgenommen (Ebene 0-4) (Adams, Kühn, Stör & Zelm, 2007):

Ebene 4 Strategische und taktische Unternehmensführung

[8]Manufacturing Execution System
[9]Enterprise Resource Planning

Ebene 3 Produktionsleitung, Qualitätsmanagement

Ebene 2 Automatisierungs- und Kontrollsysteme

Ebene 1 Automatisierungsobjekte

Ebene 0 Prozess

Die einzelnen Ebenen lassen sich auch in einer klassischen Automatisierungspyramide nach Polke und Ahrens (1994) in Abbildung 2.1.14 wiederfinden.

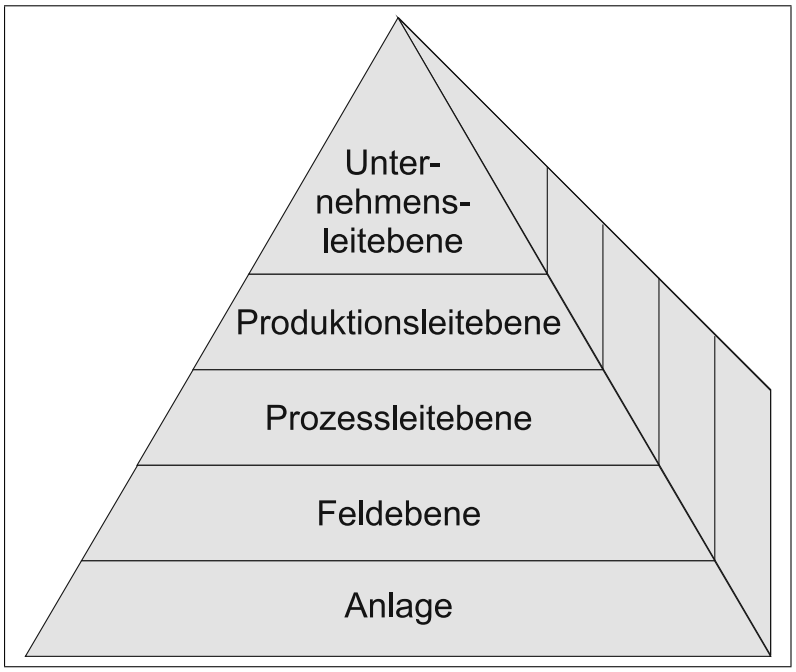


Abbildung 2.1.14: Klassische Automatisierungspyramide

Versucht man nun die Automotive-Domäne direkt auf diese Pyramidenstruktur abzubilden, so ist dies aufgrund der in erster Linie adressierten Domäne der Produktionsprozesse schwierig.

Für die Zuordnung wird an dieser Stelle eine Pyramide bestehend aus Planungsebene, Prozessleitebene, Zellebene, Feldebene und Sensor-Aktor-Ebene mit nachfolgender funktionaler Spezifikation vorausgesetzt.

Planungsebene Die Planungsebene bildet den Übergang zum kaufmännischen Bereich, in dieser werden die Bereiche Unternehmensebene und Betriebsleitebene zusammengefasst. Durch die Orientierung auf produzierende Prozesse finden sich hier produktions- und produktspezifische Funktionen wie beispielsweise Produktionsfeinplanung, Materialwirtschaft (Beschaffung, Lagerhaltung, Disposition), Verkauf und Marketing. Ebenso finden sich weitere organisatorische und verwaltungsbezogene Funktionen wie Finanz- und Rechnungswesen, Personaldatenerfassung, Personalwirtschaft, Betriebsdatenerfassung und Maschinendatenerfassung. Zudem ist die Vorgabe von Zielen für die Produktion von zentraler Bedeutung für den Prozess.

Prozessleitebene In der Prozessleitebene werden inhaltlich benachbarte Zellen zu einer technologischen Anlage zusammengefasst. Die Anlage wird als Ganzes geführt. Zu diesem Zweck erfolgen auf dieser höheren Ebene Synchronisations-, Koordinations-, Bedienungs- und Protokollierungsaufgaben, die sich auf die gesamte Anlage beziehen. Diese Aufgaben erfolgen auch in Verbindung mit einer Anlagenoptimierung.

Zellebene In der Zellebene werden Feldgeräte auf Basis ihrer inhaltlichen Beziehungen zu technischen Einheiten sogenannten Zellen zusammengefasst. Innerhalb dieser Zellen erfolgt die Zielvorgabe für die einzelnen Feldgeräte sowie deren Synchronisation und Koordination.

Feldebene In der Feldebene befinden sich die Feldgeräte, deren Aufgabe die unmittelbare Steuerung und Regelung ist. Zusätzliche Funktionen sind Anzeigen von Prozessgrößen und Informationen, Bedienung und Einstellungen im Rahmen des Prozessablaufs und Protokollieren der Informationen. Geräte in dieser Ebene können u.a. prozessnahe Speicherprogrammierbare Steuerungen (SPS), Industrie-PCs (IPC) oder auch intelligente Sensoren und Aktoren sein. Ein wichtiger Bestandteil der Feldgeräte sind analoge bzw. digitale Komponenten zur Kommunikation (z.B. Feldbusse).

Sensor-Aktor-Ebene Die Sensor-Aktor-Ebene bildet den direkten Übergang zum gesteuerten Prozess, dabei dienen die Sensoren zur Erfassung von relevanten Prozess-Informationen und die Aktoren zur Ausgabe von Prozessgrößen und somit zur Beeinflussung des Prozessablaufs.

Eine Abbildung der Ebenen ist im unteren Bereich der Pyramide, in denen die Spezifikation des Produktionsprozesses nur eingeschränkt zum Tragen kommt, direkt möglich. So können die in einem Fahrzeug vorhandenen Sensoren und Aktoren zur Erfassung und zur Ausgabe der Prozessgrößen der Sensor-Aktor-Ebene zugeordnet werden. Die einzelnen Steuergeräte lassen sich mit ihrem Aufgabenbereich Steuerung und Regelung als Feldgeräte in die Feldebene einsortieren. Die Kommunikation zwischen den einzelnen Steuergeräten erfolgt durch die automobilen Bussysteme wie LIN, CAN, FlexRay und MOST. In der Zellebene werden einzelne Feldgeräte auf Basis ihrer inhaltlichen Verbundenheit zusammengefasst. In einem Automobil können als einzelne Zellen die Bereiche Antrieb, Komfort und Entertainment identifiziert werden.

Bei den nächsten beiden Ebenen ist eine direkte Abbildung wie zuvor geschehen nicht möglich. Stattdessen soll eine interpretierende Einordnung vorgenommen werden. In der Leitebene lässt sich das Fahrzeug als Ganzes einordnen, insbesondere sofern eine Optimierung und Sollvorgabe im Rahmen einer zentralisierten Verwaltungsebene erfolgt. Als ein Beispiel für solch eine zentrale Vorgabe können die vom Fahrer auswählbaren unterschiedlichen Einstellungen für die Fahrwerkscharakteristik bei BMW[10] genannt werden.

[10] `http://www.bmw.com/com/de/insights/technology/technology_guide/articles/electronic_damper_control.html` [08.01.2015]

Geht man über die Grenzen eines Fahrzeuges hinaus können durch die Hinzunahme von Car-2-Car-Kommunikation zusätzliche Aspekte der Leitebene ergänzt werden.

Die Planungsebene kommt bei einem einzelnen Fahrzeug nicht zum Tragen. Erst in Hinblick auf eine größere Anzahl von Fahrzeugen im Sinne eines Flotten- oder Fuhrparkmanagements sowie einer übergeordneten Fahrzeugbeeinflussung lässt sich diese Ebene integrieren. Das Flottenmanagement mit seinen Aufgaben und Anforderungen bietet dabei den Übergang zur Unternehmensleitung. In diesen beiden oberen Ebenen sind TCP/IP-basierte Kommunikationsmechanismen angesiedelt.

2.2 Der Betriebssystemstandard OSEK/VDX als Beispiel für die automotive Applikationsebene

In dem vorhergehenden Abschnitt wurde die Vernetzung von Steuerung über einzelne Bussysteme und die Verknüpfung von einzelnen Bussystemen zu einem heterogenen Netzwerk betrachtet. Das Gerüst eines automotive Kommunikationsnetzwerkes ist somit vorhanden. Die architektur- und kommunikationsbezogenen Einflussfaktoren auf das Verhalten und die Eigenschaften des Gesamtsystems sind somit präsent. Im nächsten Erweiterungsschritt wird ein einzelner Netzwerkknoten, d.h. ein Steuergerät, detaillierter betrachtet. Eine spezifische Funktion oder Anwendung (ohne Einschränkung hinsichtlich der Komplexität, ggf. handelt es sich auch um mehrere Funktionen) wird auf einem solchen Netzwerkknoten realisiert. Zumeist übernimmt ein spezifisches Betriebssystem die Ressourcenverwaltung. Diese Ressourcenverwaltung hat konsequenterweise auch einen Einfluss auf das Verhalten des Gesamtsystems. Im Automobilbereich existiert für Betriebssysteme der Standard OSEK OS und OSEKtime OS (eine Berücksichtigung von Betriebssystemen findet sich ebenso in AUTOSAR).

Das Gremium OSEK-VDX[11] besteht aus unterschiedlichen Fahrzeugherstellern, Zulieferern sowie Steuergeräte- und Software-Herstellern. Es ist ein Zusammenschluss des 1993 gegründeten Gremiums „**O**ffene **S**ysteme und deren Schnittstellen für die **E**lektronik im **K**raftfahrzeug“ (kurz OSEK) und der französischen Initiative „**V**ehicle **D**istributed E**x**ecutive“ (kurz VDX). Das zentrale Ziel war die Entwicklung eines Betriebssystemstandards als hardware- und netzwerkunabhängige Schnittstelle, dies soll bei Anwendungen im Segment Automotive die Portabilität und Wiederverwendbarkeit erhöhen.

Ergänzend wurden Standardisierungen für ein Echtzeitbetriebssystem, das Netzwerkmanagement und die Kommunikation entwickelt. Die resultiert in verschiedenen Standards. Hierzu zählen ein prioritätsgesteuertes Echtzeitbetriebssystem (OSEK OS - OSEK Operating System) sowie die zugeordnete Kommunikation (OSEK COM - OSEK Communication) und das Netzwerkmanagement (OSEK NM - OSEK Network Management). Ergänzend dazu gibt es eine Beschreibungssprache (OSEK OIL - OSEK Implementation Language) sowie ein Debug-Interface (OSEK ORTI - OSEK Runtime Interface). Zudem erfolgte mit OSEKtime OS die Standardisierung eines zeitgesteuerten Betriebssystems mit einer fehlertoleranten Kommunikation (OSEK FTCOM - OSEK Fault Tolerant Communication). (Zimmermann & Schmidgall, 2011;

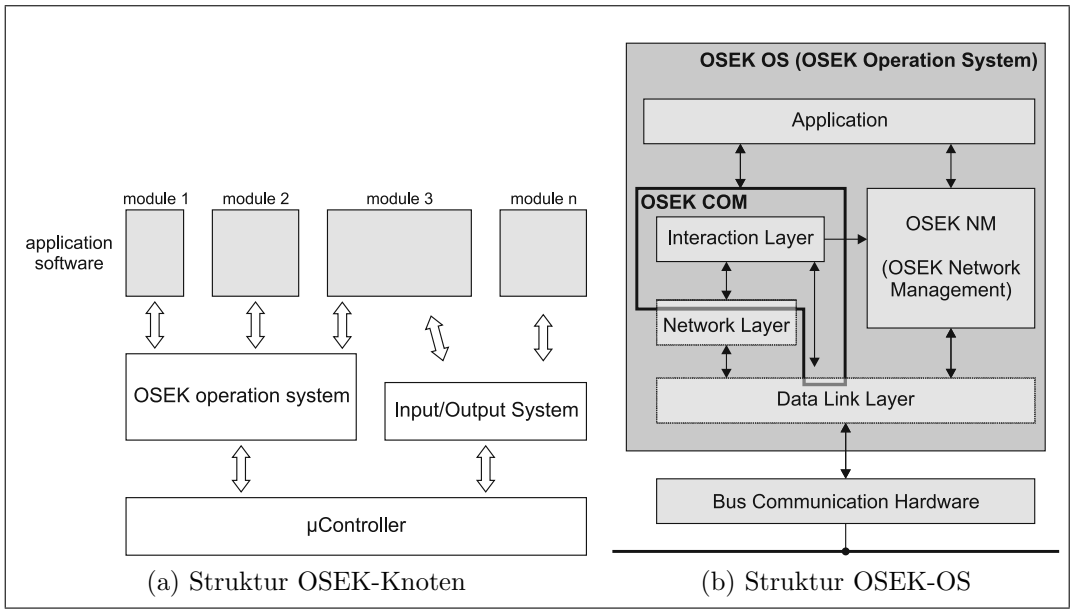
[11] `http://www.osek-vdx.org/` [08.01.2015]

Homann, 2005)

Zur Vorbereitung der Modellierung von Betriebssystem-Funktionalitäten zur exemplarischen Erweiterung der Modellierung auf Anwendungsebene werden an dieser Stelle als Referenz die Betriebssystem-Standards OSEK-OS (OSEK/VDX, 2005) und OSEKtime-OS (OSEK/VDX, 2001b) mit der jeweils zugeordneten Kommunikation OSEK-COM (OSEK/VDX, 2004) und OSEK-FTCOM (OSEK/VDX, 2001a) basierend auf den zugehörigen Spezifikationen sowie (Zimmermann & Schmidgall, 2011; Homann, 2005) kurz eingeführt.

2.2.1 Prioritätsgesteuerter Betriebssystem-Standard OSEK-OS

Betriebssysteme welche den OSEK-OS Standard erfüllen eignen sich zur Realisierung von ereignisbasierten Steuerungssystemen. Ein Schwerpunkt des Standards liegt auf der Portierbarkeit einzelner Anwendungsmodule. Dieses ermöglicht einen hohen Grad an Flexibilität und Modularität. In Verbindung mit diversen Scheduling-Mechanismen und der Möglichkeit zur Konfiguration ist der Betrieb von OSEK OS für unterschiedliche Anwendungen und Hardwarekomponenten möglich. Über definierte „Conformance Classes“ wird spezifiziert, welchen Funktionsumfang die Realisierung eines OSEK-OS konformen Betriebssystems besitzt.



(a) Struktur OSEK-Knoten (b) Struktur OSEK-OS

Abbildung 2.2.1: Struktur eines OSEK-Knotens (OSEK/VDX, 2005) und OSEK-OS-Struktur (OSEK/VDX, 2004)

Die definierten Softwareschnittstellen innerhalb eines Steuergeräts und die Struktur des Betriebssystems sind in Abbildung 2.2.1 dargestellt, hier findet sich zum einen die Knotenstruktur aber auch die Struktur des Betriebssystems an sich. Zur Realisierung der Portierbarkeit und Wiederverwendbarkeit sind Dienste und Funktionen für die Schnittstelle zwischen Anwendung und Betriebssystem standardisiert (Portabilität auf Quellcode-Ebene). Die Hardware-Funktionen können über die Betriebssystem-

dienste und -funktionen verwendet werden. Nachfolgend werden das Task-Modell und das Scheduling aufgeführt. Zusätzlich sind im Anhang A.3.1 ergänzend ausgewählte Dienste und Funktionen kurz aufgeführt.

2.2.1.1 Task und Scheduling

Das Betriebssystem steuert die Ausführung von zwei Typen von Objekten: Interrupt-Service-Routinen (ISR) und Tasks. Diese können mit Hilfe des Betriebssystems auf den Prozessor und weitere Module zugreifen. Die Ausführung beinhaltet drei Prioritätsebenen: die Interrupt-Ebene, die logische Ebene für Scheduling-Aufgaben und die Task-Ebene. Die Task-Ebene untergliedert sich wiederum in *non-preemptives*, *full preemptives* sowie *mixed preemptives* Scheduling. Die Prioritäten werden durch den Entwickler zugeordnet.

Die einzelnen Tasks werden in zwei Gruppen eingeordnet: Basic Task und Extended Task. Erstere können drei Zustände annehmen (running, suspended und ready). In Abbildung 2.2.2 umfasst das Zustandsmodell der Basic Task die ungefüllten Zustände, das Zustandsmodell des Extended Task umfasst zusätzlich den gefüllten Zustand. Basic Tasks geben den Prozessor in drei Fällen frei. Erstens sie wurden beendet (terminate), es wird in einen Task mit höherer Priorität (preempt) oder über einen Interrupt wird in eine ISR gewechselt. Der zusätzliche Zustand waiting bei den Extended Tasks wird eingenommen, wenn ein Task auf eine Ressource oder ein Ereignis wartet. Das Betriebssystem versetzt den Task in den Zustand *ready*, sobald die Ressource oder das Ereignis verfügbar ist. Dies führt zu einer besseren Ausnutzung der Prozessorleistung. Besitzen im Zustand *ready* mehrere Tasks eine identische Priorität, so werden diese nach der Reihenfolge ihrer Aktivierung gestartet. Aus Effizienzgründen wird eine dynamische Prioritätenverwaltung von OSEK nicht unterstützt. Mehrfachaktivierungen werden durch den Scheduler verwaltet und deren Anzahl wird mit Hilfe eines Task-Attributes festgelegt.

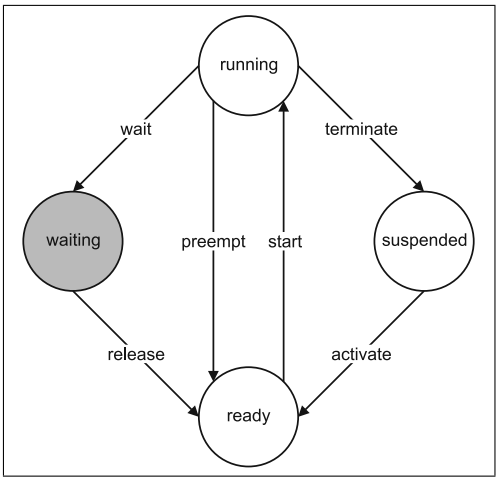


Abbildung 2.2.2: OSEK-OS Taskmodell (OSEK/VDX, 2005)

Innerhalb des full preemptiven Scheduling kann ein Task durch ein Ereignis die Kontrolle über den Prozessor verlieren. In diesem Fall werden Prozessorregister, Stack- und Programmzeiger gesichert. Bei einem solchen Kontextwechsel werden zusätzliche Ressourcen benötigt und der Synchronisationsaufwand ist höher. Daher müssen kritische Codestellen mittels Systemfunktionen geschützt werden.

Bei der Verwendung des non-preemptiven Scheduling, können sich die einzelnen Tasks nicht verdrängen. So können einzelne Tasks zu Gruppen zusammengefasst werden. Die Gruppenmitglieder können sich gegenseitig nicht verdrängen und lediglich von Tasks anderer Gruppen mit einer höheren Priorität verdrängt werden. Bei dem *mixed preemptive Scheduling* befinden sich preemptive und non-preemptive Tasks innerhalb eines Systems.

2.2.1.2 Kommunikation OSEK-COM

Die Kommunikation wird gesondert in dem Standard OSEK-COM betrachtet. Dieser stellt Dienste für die Datenkommunikation zur Verfügung. Zur Steigerung von Portabilität, Wiederverwendbarkeit und Interoperabilität werden Informationen über die Lokalität (interne oder externe Kommunikation) vor dem Aufrufer versteckt. Der Kommunikationsblock hat, wie in Abbildung 2.2.1 zu erkennen, einen dreischichtigen Aufbau: Interaction Layer (IL), Network Layer (NL) und Data Link Layer (DL). Der Interaction Layer beinhaltet Dienste, welche eine Anwendung zur Kommunikation nutzt. Externe Anforderungen werden zur untergeordneten Schicht weitergereicht und die interne Kommunikation wird innerhalb der Schicht abgearbeitet. Der untergeordnete Network Layer hängt vom externen Bussystem ab. Im Rahmen dieser Schicht erfolgt die Segmentierung und De-Segmentierung von Nachrichten, das Versenden von Bestätigungen und Kontrollflussmechanismen. Der Data Link Layer wird ebenso wie der Network Layer in OSEK COM nicht genau spezifiziert.

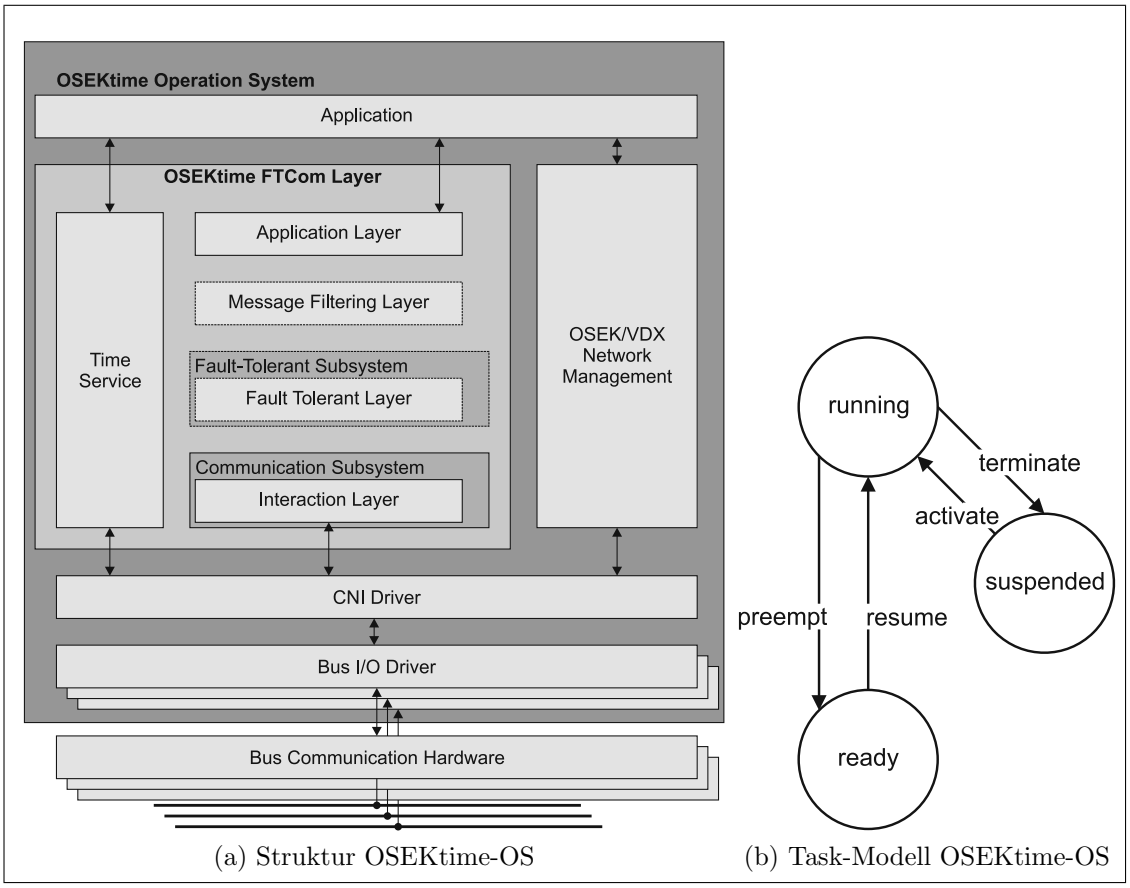
Die Kommunikation zwischen Tasks und ISRs basiert auf Nachrichten (Messages), die durch einen Identifier unterschieden werden. Interne Kommunikationen zeichnen sich dadurch aus, dass der IL die gesendeten Daten sofort dem Empfänger bereitstellt. Im Falle einer externen Kommunikation wird die Nachricht verpackt (I-PDU - Interaction Layer Protocol Data Units) und an die unterliegende Schicht weitergeleitet. Beim Empfang von Nachrichten wird zwischen „queued“ und „unqueued“ differenziert.

2.2.2 Zeitgesteuerter Betriebssystem-Standard OSEKtime-OS

Das zeitgesteuerte Betriebssystem übernimmt das Management von Ressourcen und Zeit sowie das Scheduling der Tasks. Zudem kontrolliert es die Anwendungssoftware und die Kommunikation. Der Aufbau ist in Abbildung 2.2.3a dargestellt. Die Schicht FTCOM (Fault Tolerant Communication) übernimmt die Kommunikation zwischen einzelnen Knoten, die Fehlererkennung und die fehlertoleranten Funktionen innerhalb des Kommunikationssubsystems. Das OSEK Network Management beinhaltet globale und lokale Verwaltungsmethoden.

Auch hier existieren zwei Prozesstypen, die Interrupt Service Routinen und die Tasks mit zwei unterschiedlichen Abarbeitungsebenen, die Interrupt-Ebene und die ttTask-Ebene (time triggered). Zusätzlich kann als Subsystem ein OSEK OS integriert werden. Dieses kann parallel erfolgen oder die Ausführung erfolgt innerhalb des sogenannten ttIdleTask, abhängig von der Hardware.

Nachfolgend werden das Task-Modell und das Scheduling aufgeführt. Im Anhang A.3.2 finden sich ergänzend ausgewählte Dienste und Funktionen.



(a) Struktur OSEKtime-OS (b) Task-Modell OSEKtime-OS

Abbildung 2.2.3: OSEKtime-OS-Struktur und Task-Modell (OSEK/VDX, 2001b)

2.2.2.1 Task und Scheduling

In der Abarbeitung ist das Blockieren von Tasks verboten. Zur Gewährleistung der Echtzeiteigenschaften bei der sequentiellen Abarbeitung muss die maximale Ausführungszeit (WCET - Worst Case Execution Time) bekannt sein. Die einzelnen Tasks werden jeweils durch einen vom Dispatcher erzeugten Aktivierungsevent gestartet.

Das Taskmodell in Abbildung 2.2.3a beinhaltet drei Zustände: running, suspended und preempted. Der Zeitpunkt der Aktivierung eines Task und alle weiteren Eigenschaften (z.B. Deadline, Startzeit, maximale Laufzeit) stehen in der Dispatcher-Tabelle. Ein aktivierter OSEKtime-OS Task wechselt im Gegensatz zu einem OSEK-OS Task direkt in den Zustand running, verdrängt den aktiven Task und kann ausschließlich von sich selbst beendet werden. Die Mehrfachaktivierung eines Tasks ist möglich.

Das Scheduling ist ein statisch, preemptives Verfahren, bei dem jeder zeitgesteuerte Task einen anderen verdrängen kann. Allerdings sind keine Blockierungen oder Ereignisse erlaubt.

Der Dispatcher startet die Tasks auf Basis einer statisch vorgegebenen Reihenfolge (Dispatcher-Tabelle) ohne Prioritäten. Ist bereits ein Task aktiv, wird dieser bis zur Beendigung des neu aktivierten Tasks verdrängt. Bei der Abarbeitung des vorgegebenen Ablaufs (Dispatcher-Tabelle) darf sich bei dem Stack-orientierten Scheduling

keine Verletzung der Deadlines ergeben und kein Überlauf des Stacks erfolgen.

Während der Entwicklungsphase übernimmt der Dispatcher die Überwachung der Deadlines sowie eine Fehlerbehandlung bei deren Überschreiten, diese Funktionalität ist in der Betriebsphase nicht vorhanden.

Eine spezielle Rolle nimmt der ttIdleTask ein, dieser kann nicht beendet werden und wechselt in den Zustand running, wenn kein anderer Task aktiviert ist. Zu Beginn und Ende einer Dispatcher-Runde kann ausschließlich der ttIdleTask aktiv sein, dieser ist zudem nicht in der Dispatcher-Tabelle festgehalten.

2.2.2.2 Kommunikation

Ein wichtiger Bestandteil des OSEKtime-Systems ist die fehlertolerante Kommunikation (OSEK-FTCOM Spezifikation) mit fehlertoleranter Kommunikationsschicht, Echtzeitprotokollen und -systemen. Die standardisierten Schnittstellen für Dienste und Funktionen zur Kommunikation umfassen eine globale Nachrichtenbehandlung, einen Start-up und eine Reintegrationsunterstützung und einen Dienst zur Synchronisation mit externen Zeitgebern.

Die einzelnen Schichten des Kommunikationssystems sind in Abbildung 2.2.3a erkennbar. Zu diesen zählen Application Layer (Anwendungsschnittstelle), Message Filtering Layer, Fault Tolerant Layer (Dienste für fehlertolerante Funktionen) und Interaction Layer sowie CNI-Driver (Communication Network Interface) als Schnittstelle zum Netzwerk

Die Nachrichtenschnittstelle innerhalb der FTCOM-Schicht ist angelehnt an den Transfer von „unqueued“-Nachrichten. Die Kommunikation (Datenaustausch) zwischen Task erfolgt über die FTCOM-API. Der Anwendung bleibt verborgen ob der interne oder ein externer Kommunikationsweg gewählt wurde.

2.3 Modellierung von Kommunikationsarchitekturen

Die Modellierung von Kommunikationsarchitekturen basiert im Wesentlichen auf den Kommunikationsprotokollen, deren Spezifikation informal, formal oder gemischt erfolgt. Im Allgemeinen sind die Spezifikationen oft nicht vollständig, so dass bei einer Implementierung gewisse zusätzliche Annahmen getroffen werden müssen. Eine direkte Ableitung, d.h. eine automatische Generierung, ist daher oft nicht möglich. Für formale oder gemischte Spezifikationen werden zur Standardisierung von Protokollen häufig standardisierte Formal Description Techniques (FDT) verwendet. Bekannte Vertreter solcher FDTs zur Beschreibung von verteilten Systemen sind zum Beispiel ESTELLE, SDL (Specification and Description Language) und LOTOS. SDL ist weit verbreitet und es existiert eine Vielzahl von Tools zur Unterstützung bei Entwicklung und Verifikation. Bei informalen Spezifikationen gibt es das Problem, dass diese nicht eindeutig oder widersprüchlich sind und die Verifikation schwierig ist.

Zur Vorbereitung der Entwicklung eines Ansatzes zur Modellierung und Umsetzung von Modellen für die Kommunikation im Automobil werden einführend einige Grundlagen im Bereich der Modellierung auszugsweise betrachtet. Begonnen wird mit einer allgemeinen Betrachtung der Discrete-Event-Modellierung. Anschließend werden aus-

gewählte Typen von Modellen vorgestellt. Berücksichtigt werden hier Petri-Netze, einfache Zustandsmodelle und erweiterte Zustandsmodelle (ESTELLE und SDL) sowie die Multi-Domänen Modellierung mit ausgewählten Tools (Ptolemy, MLDesigner, MATLAB/Simulink und LabVIEW).

2.3.1 Discrete-Event-Modellierung

Für den Begriff System finden sich unterschiedliche Definitionen. In (Cassandras & Lafortune, 2008) wird zur Klärung der Bedeutung unter anderem das IEEE Standard Dictionary of Electrical and Electronic Terms folgendermaßen zitiert.

> „A system is a combination of components that act together to perform a function not possible with any of the individual parts. [IEEE Standard Dictionary of Electrical and Electronic Terms]“

Von wesentlicher Bedeutung sind dabei die Aspekte, dass sich ein System aus verschiedenen interagierenden Komponenten zusammensetzt, welche gemeinsam eine Funktion realisieren.

Für ein Discrete-Event-System gilt, dass dieses einen diskreten Zustandsraum besitzt. Änderungen des Zustandes sind lediglich zu diskreten Zeitpunkten möglich, dies ist gleichbedeutend mit dem Auftreten von diskreten Ereignissen. Die Abfolge von Zuständen (Zustandstrajektorie) wird durch das Auftreten von Ereignissen bestimmt (asynchron). Eine formale Definition der Spezifikation eines klassischen Discrete Event Systems erfolgt in (Zeigler, Praehofer & Kim, 2000) durch das folgende Tupel:

$$M = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

für dieses gilt:

X ist die Menge von Eingangswerten

S ist die Menge von Zuständen

Y ist die Menge von Ausgabewerten

$\delta_{int} : S \to S$ ist die interne Übergangsfunktion

$\delta_{ext} : X \times S \to S$ ist die externe Übergangsfunktion,
mit $Q = \{(s, e) | s \in S, 0 \leq e \leq ta(s)\}$ als Menge aller Zustände und e zeitlicher Abstand zur letzten Transition

$\lambda : S \to Y$ ist die Ausgabefunktion

$ta : S \to \Re^{+}_{0,\infty}$ ist die Menge der positiven Zahlen inklusive 0 und ∞

Zudem findet sich als Erweiterung die parallele Discrete Event Systeme, dabei wird ein einfaches System durch das folgende Tupel beschrieben:

$$M = \langle X_M, Y_M, S, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta \rangle$$

für dieses gilt:

$X_M = \{(p, v)|p \in IPorts, v \in X_p\}$ ist die Menge von Eingangsports und Eingangswerten

$Y_M = \{(p, v)|p \in OPorts, v \in Y_p\}$ ist die Menge von Ausgangsports und Ausgangswerten

S ist eine Menge von sequentiellen Zuständen

$\delta_{int} : S \to S$ ist die interne Zustands-Übergangsfunktion

$\delta_{ext} : Q \times X_M^b \to S$ ist die externe Zustands-Übergangsfunktion

$\delta_{con} : Q \times X_M^b \to S$ ist die konfluente Übergangsfunktion

$\lambda : S \to Y^b$ ist die Ausgabefunktion

$ta : S \to \Re_{0,\infty}^+$ als Zeitfunktion

$Q = \{(s, e)|s \in S, 0 \leq e \leq ta(s)\}$ als Menge aller Zustände

In einem Modell werden ausgewählte Eigenschaften eines Systems oder Teilsystems beschrieben. Durch die Auswahl der Eigenschaften erfolgt eine Abstraktion der realen Daten, Methoden und Funktionen eines Systems oder Teilsystems.

Ein hierarchisches Modell wird als ein Modell definiert, welches aus verknüpften Komponenten besteht. Eine einzelne Komponente kann dabei entweder atomar sein, oder ihrerseits aus einem gekoppelten Modell bestehen.

2.3.2 Petri-Netze

Petri Netze (PN) sind ein auf der Arbeit von Carl Adam Petri basierendes formales Modell bestehend aus Plätzen und Transitionen. (Priese & Wimmel, 2008; Wimmel, 2008; Cassandras & Lafortune, 2008) Es gibt eine Vielzahl von unterschiedlichen Varianten. In einer einfachen Variante kann dieses Platz-Transitions-Netz als 3-Tupel $PN = \langle P, T, F\rangle$ definiert werden mit

P Menge der Plätze

T Menge der Transitionen

$F \subseteq (P \times T) \cup (T \times P)$ Flussrelation oder Kantenfunktion

Dieses wird zumeist um eine Anfangsmakierung $m0$ ergänzt. Häufig findet sich auch eine Darstellung als 4-Tupel $PN = \langle P, T, \mathcal{F}, \mathcal{B}\rangle$, welche die Flussrelation oder Kantenfunktion in eine Forward-Matrix ($|P| \times |T| - Matrix\mathcal{F}$ über N) und Backward-Matrix ($|P| \times |T| - Matrix\mathcal{B}$ über N) aufteilen. Die Pfeile werden dabei jeweils aus der Sicht des Platzes gesehen. (Priese & Wimmel, 2008). In (Zeigler et al., 2000) wird ein entsprechendes 5-Tupel definiert, welches zusätzlich die Anfangsmarkierung berücksichtigt. In einem ersten Erweiterungsschritt können Plätzen jeweils Kapazitäten

oder Kanten jeweils Gewichte zugeordnet werden. Mit Hilfe der Marken lässt sich das dynamische Verhalten eines Systems modellieren.

Aufgrund ihrer Semantik eignen sich Petri Netze insbesondere zur Modellierung von Wechselwirkungen zwischen parallelen Prozessen. Somit können sie zur Beschreibung von Protokollen (verhaltens- und kommunikationsorientiert) verwendet werden. Die Abläufe der Kommunikation werden durch den Markenfluss anschaulich visualisiert. Mit Hilfe der Petri Netze lassen sich wichtige Eigenschaften (Sicherheit, Konflikt- und Verklemmungsfreiheit) von Kommunikationsabläufen nachweisen. Dabei legen die Struktur und die Anfangsmakierung des Petri-Netzes die erreichbaren Zustände (erreichbaren Markierungen) fest.

Ein Vorteil von Petri Netzen sind die grafische Entwicklung und die intuitive Notationen in Verbindung mit der formalen Mächtigkeit zur Analyse. Gleichzeitig ist allerdings die Abbildung von komplexen Systemen schwierig und bei großen Systemen leidet die Übersichtlichkeit. Es gibt eine Vielzahl von Variationen von Petri Netzen. Durch Erweiterungen, welche beispielsweise Konzepte zur Strukturierung und Unterscheidung von Marken beinhalten, wird eine Reduzierung der Komplexität der grafischen Darstellung erreicht. Somit können mit Hierarchischen-PN oder Coloured-PN auch komplexe Systeme dargestellt werden. Eine tiefergehende Auseinandersetzung mit höheren Petri-Netzen insbesondere hierarchischen gefärbten Netzen findet sich in (Al Ali, 2011; Knorr, 1994).

Die Semantik der Petri-Netze findet sich auch teilweise in den Aktivitätsdiagrammen der UML wieder.

Beispiele zur Modellierung mit Petri-Netzen finden sich unter anderem in (Bago, Perić & Marijan, 2008), hier werden gefärbte zeitbehaftete Petri-Netze zur Modellierung des Controller Area Network (CAN) eingesetzt und in (Lakos, Lamp, Keen & Marriott, 1995) werden Object Petri-Netze zur Modellierung von Netzwerk Protokollen genutzt.

2.3.3 Finite-State-Machines / Statecharts

Bei der Modellierung des Verhaltens von Systemen haben sich Zustandsautomaten (State Machines) als geeignete Methode herausgestellt, welchen ein mathematisches Modell zu Grunde liegt. (Cassandras & Lafortune, 2008; Koshy, 2004; Hopcroft & Ullman, 1990)

Der endliche Automat (finite state automaton) wird in (Hopcroft & Ullman, 1990) und (Koshy, 2004) dabei als 5-Tupel $FSA = \langle \sum, Q, q0, F, \delta \rangle$ mit

$\sum$ ist das Eingangsalphabet

Q ist die endliche Menge der Zustände

$q0 \in Q$ ist der Startzustand

$F \subseteq Q$ ist die Menge der Endzustände (akzeptierende Zustände)

$\delta : Q \times \sum \rightarrow Q$ ist die Übergangsfunktion

definiert und bildet eine Grundlage für weitere Modelle und die zustandsbasierte Modellierung. So berücksichtigen weitere Modelle die Definition von Ausgabefunktionen. In (Koshy, 2004) werden diese als finite-state machine (FSM) bezeichnet und als 6-Tupel $FSM = \langle S, I, O, f, g, s0 \rangle$ mit

S ist die endliche Menge der Zustände

I ist das Eingangsalphabet

O ist das Ausgabealphabet

$f : S \times I \to S$ ist die Übergangsfunktion

$g : S \times I \to O$ ist die Ausgabefunktion

$s0 \in S$ ist der Startzustand

definiert. Da die Ausgabefunktion sowohl von Zustand als auch von Eingabe abhängig ist handelt es sich um einen Mealy-Automaten. Im Gegensatz dazu ist bei einem Moore-Automaten die Ausgabe nur abhängig vom aktuellen Zustand. (vgl. Cassandras & Lafortune, 2008)

Für Systeme, die sich mit Zustandsautomaten modellieren lassen, können spezifische Eigenschaften abgeleitet werden:

- Das System lässt sich durch eine endliche Anzahl von Zuständen beschreiben.
- Es gibt eine endliche Menge von Eingaben oder Ereignissen als Trigger von Zustandsübergängen.
- Das Systemverhalten basiert auf dem aktuellen Zustand und der auftretenden Eingaben bzw. dem auftretenden Ereignis.
- Es gibt einen initialen Systemzustand.
- Für jeden Zustand ist das Verhalten für alle möglichen Eingaben oder Ereignisse definiert.

Ein großer Vorteil von Zustandsautomaten ist die intuitive Notation sowie die grafische Entwicklung in Verbindung mit einer guten Toolunterstützung. Insbesondere bei großen und komplexen Systemen ergeben sich massive Probleme in den Bereichen Übersichtlichkeit und Zustandsraumexplosion, welche die Systemanalyse und Beherrschbarkeit einschränken.(Cassandras & Lafortune, 2008)

Für die Darstellung von komplexen Systemen eignen sich beispielsweise EFSMs (Extended FSM) oder auch CFSMs (Communicating FSM) sowie Statecharts. Statecharts sind ein von David Harel (Harel, 1986) erweitertes Automatenmodell, in dem Hierarchie, Zustandshistorie, bedingte Zustandsübergänge, synchrone Kommunikation und Nebenläufigkeit unterstützt werden. Eine leichte Adaption dieser findet sich in der UML (Rupp, Queins & Zengler, 2007).

In (Negri, Sami, Macii & Terranegra, 2004) werden FSMs zur Modellierung des Energieverbrauchs eines Bluetooth Moduls genutzt.

2.3.4 Erweitertes Zustands-Transitionsmodell

Das Modell der Extended Finite State Machines kombiniert endliche Automaten mit der Leistungsfähigkeit von höheren Programmiersprachen. Die endlichen Automaten werden dabei um Variablen erweitert. Diese können Eingaben, Ausgaben und lokale Variablen für die interne Verarbeitung sein. Wichtige Beispiele für EFSM sind die Spezifikationssprachen SDL und Estelle.

2.3.4.1 Specification and Description Language - SDL

Die Spezifikationssprache SDL ist ein Standard der ITU-T (International Telecommunication Unit Telecommunication Standardization Sector) zur Beschreibung von (verteilten) Systemen im Telekommunikationsbereich (ITU-T, 2002). Sie basiert auf dem Modell der EFSM (Extended Finite State Machines). Die Zustandsmaschinen werden parallel abgearbeitet und kommunizieren über Signale miteinander. (Hogrefe, 1989)

In Standardisierungsprozessen wird die SDL zur Beschreibung von Architektur, Verhalten, Daten und dem statischen Interface verwendet. So wird sie auch bei der Spezifikation des FlexRay Standards eingesetzt, welcher eine Kombination aus Textdokument und der Spezifikations- und Beschreibungssprache SDL ist.

Ein System in SDL besteht aus einer hierarchischen Blockstruktur, welche über Kanäle verbunden sind. Die einzelnen Blöcke können hierbei einen hierarchischen Aufbau besitzen und aus kommunizierenden Blöcken oder Prozessen bestehen. Ein Prozess bildet in der SDL das Basiselement. Das Prozessverhalten wird dabei durch eine EFSM beschrieben.

Zur Unterstützung der Entwicklung und Verifikation von Systemen mittels SDL gibt es eine Vielzahl von Tools. Die Verifikation erfolgt dabei mit Petri-Netzen oder FSMs. Sie hat ihre Stärken in der Verhaltensbeschreibung bis hin zur direkten Implementierung. Zur Beschreibung von Anforderungen ist sie weniger geeignet.

Weit verbreitet ist die Verwendung der SDL im Bereich der drahtlosen Kommunikation. Beispielsweise wird in (Showk et al., 2009) die SDL zur Modellierung des Long Term Evolution (LTE) Protokolls verwendet. Auch für den Vorgänger Universal Mobile Telecommunications System (UMTS) finden sich Modelle in SDL (Alvarez et al., 2006). Aber nicht nur für die mobilen Kommunikationssysteme wird SDL verwendet, in (Zaghal & Khan, 2005) wird das Transmission Control Protocol (TCP) modelliert.

2.3.4.2 Extended State Transition Language - Estelle

Estelle ist eine von der ISO standardisierte formale Beschreibungstechnik (FDT) zur Spezifikation von verteilten Systemen und Kommunikationsprotokollen und -diensten. Ebenso wie SDL basiert sie dabei auf EFSMs. Die Erweiterung der Finite State Machines umfasst dabei unter anderem das Handling von Variablen. Zudem können Transitionen unabhängig von Eingangssignalen sein und mit einer Zeitverzögerung behaftet sein. (Hogrefe, 1989)

Die Spezifikation der Systeme erfolgt in Estelle hierarchisch. Die Hauptelemente bilden Module, welche aus Submodulen bestehen können und über bidirektionale

Kanäle miteinander Nachrichten austauschen. Ein Modul kann die ihm zugeordneten Variablen verarbeiten und kann Ausgangssignale erzeugen. Das Modul-Verhalten wird mit EFSMs beschrieben. Ein Modul kann dabei mehrere Schnittstellen (Interaktionspunkte) besitzen. Diesen Schnittstellen können jeweils FIFO-Queues zugeordnet werden.

Beispiele zur Verwendung von Estelle finden sich zum Beispiel bei der Modellierung des Asynchronous Transfer Mode (ATM) Signaling Protocol in (Tasak, 1997) und des ATM available bit rate (ABR) control protocol in (W. Huang, 2001).

2.3.5 Multi-Domänen Modellierung und Simulation

Viele Systeme und Sachverhalte lassen sich nicht mit einem Formalismus abbilden. Häufig ist eine kombinierte diskrete und kontinuierliche Modellierung und Simulation notwendig. Daher gibt es in sogenannten Multidomänen Modellierungs- und Simulationswerkzeugen die Möglichkeit verschiedene Berechnungsmodelle zu kombinieren.

Ein Referenztool ist Ptolemy[12], ein Projekt der Universität Berkeley, in welchem eine Methodik umgesetzt wird, die eine Kombination verschiedener Berechnungsmodelle innerhalb einer Simulation erlaubt. Die Interaktion zwischen den Berechnungsmodellen wird im Simulationskern durch eine Anpassungsschicht gewährleistet. Somit sind Modelle unterschiedlicher Simulationsdomänen zu einem simulationsfähigen Gesamtmodell integrierbar. Besonders relevant sind die Domänen DE (Discrete Event), FSM (Finite State Machine) und SDF (synchroner Datenfluss).

2.3.5.1 Der Ptolemy-Ansatz mit Ptolemy Classic und Ptolemy II

Ptolemy setzt sich thematisch mit der Modellierung, der Simulation und dem Design von eingebetteten Echtzeitsystemen auseinander. Der Hauptbetrachtungspunkt ist dabei eine domänenübergreifende Entwurfs- und Modellierungsmethode. Hierbei geht es insbesondere um die Kombination von unterschiedlichen Berechnungsmodellen (Models of Computation), welche die Steuerung der Interaktionen zwischen Komponenten innerhalb eines Modells beschreiben. (Eker et al., 2003; Lee et al., 2001)

Aus diesem Projekt heraus sind Tools zur Modellierung und Simulation erwachsen. Dies sind zum einen Ptolemy Classic, welches auf Grund diverser Einschränkungen lediglich noch als experimentelle Plattform verwendet wird, und zum anderen Ptolemy II, welches die Classic-Variante abgelöst hat.

2.3.5.1.1 Ptolemy Classic Ptolemy Classic (1990)[13] unterstützt als heterogenes Simulations- und Entwurfswerkzeug unterschiedliche Models of Computation wie zum Beispiel Datenfluss, Discrete Event und Finite-State Machines. Die hierarchische und attributierte grafische Erstellung von Modellen mit Komponenten aus unterschiedlichen Domänen sowie die simulationsbasierte Modellanalyse wird unterstützt. Die Umsetzung von Ptolemy Classic erfolgte in C++. (Buck, Ha, Lee & Messerschmitt, 1991)

[12] `http://ptolemy.eecs.berkeley.edu/` [08.01.2015]
[13] `http://ptolemy.eecs.berkeley.edu/ptolemyclassic/` [08.01.2015]

2.3.5.1.2 Ptolemy II Ein großer Vorteil des Nachfolgers Ptolemy II (1996)[14] ist die Plattformunabhängigkeit (Java Realisierung) und das verbesserte grafische Nutzerinterface. In Ptolemy II werden die Begriffe „actor-oriented design“ und „director“ eingeführt. Als Actor werden hierbei konkurrierende Software-Komponenten verstanden, die in einem hierarchischen Modell über Ports miteinander kommunizieren können. Ein „Director“ ist die Umsetzung eines Models of Computation und ist somit für die Steuerung der Interaktion der einzelnen Komponenten verantwortlich. Eine hierarchische Kombination dieser „directors“ ist möglich. Innerhalb des Ptolemy Projektes gibt es solche „director“ für die Berechnungsmodelle: process networks (PN), discrete-events (DE), synchronous dataflow (SDF), synchronous/reactive(SR), rendezvous-based models, 3-D visualization und continuous-time.

Das Anwendungsgebiet von Ptolemy ist vielfältig. Baldwin, Kohli, Lee, Liu und Zhao (2004) und Rosello, Portilla, Krasteva und Riesgo (2009) verwenden Ptolemy zur Modellierung von drahtlosen Sensor Netzwerken.

2.3.5.2 MATLAB/Simulink

Die Toolsuite MATLAB/Simulink von Mathworks[15] ist ursprünglich ein rein numerisches Simulationssystem und verwendet im Wesentlichen ein synchrones Datenflussmodell. Durch viele Erweiterungen hat es sich allerdings zu einem Allroundwerkzeug entwickelt. Insbesondere durch Module zur eventbasierten Modellierung lassen sich komplexe Systeme realisieren und Discrete-Event Simulationen durchführen. (Angermann, Beuschel, Rau & Wohlfahrt, 2007)

Die Stärke von MATLAB/Simulink ist dabei die hohe Verfügbarkeit leistungsfähiger Codegeneratoren für Mikrocontroller und programmierbare Hardware sowie die breite Verwendung in der Industrie und die leichte Anbindung an externe Tools. Dies ermöglicht innerhalb des Systementwurfes aus Modellen direkte Implementierungen abzuleiten. Eine große Verwendung findet MATLAB/Simulink im Bereich Rapid Prototyping.

Ein Nachteil bei der Simulation von Modellen ist die Notwendigkeit der Festlegung einer Basistaktrate, die dazu führt, dass alle Komponenten in einem System mit dieser Zeitbasis simuliert werden. Bei der zeitlichen Abstraktion unterliegt man somit gewissen Beschränkungen bei der Kombination von Komponenten unterschiedlichen Detaillierungsgrades in einem Systemmodell. In den Beispielen zum MATLAB Modul SimEvents findet sich als Referenz zur Modellierung von Kommunikationsprotokollen ein Beispiel zu Ethernet LAN (mathworks, 2014).

[14] http://ptolemy.eecs.berkeley.edu/ptolemyII/ [08.01.2015]
[15] `http://www.mathworks.de/` [08.01.2015]

2.3.5.3 MLDesigner

2.3.5.3.1 Allgemein Der MLDesigner der Firma MLDesign Technologies, Inc.[16] ist eine grafische Entwicklungsumgebung zur Modellierung und Simulation von Systemen. Das Tool unterstützt die Multi-Domänen Modellierung und baut auf dem Konzept von Ptolemy auf. Im Vergleich zu Ptolemy Classic werden einige Erweiterungen eingeführt. Hierzu zählen u.a. die Erweiterung der Modellbibliothek die Einführung weiterer Domänen (z.B. die Domäne Continuous Time - Discrete Event für die Modellierung hybrider Systeme). Der Prozess der Systementwicklung soll durch eine begleitende Modellierung beginnend in frühen Entwurfsphasen bis hin zur Implementierung verbessert werden. Das Werkzeug eignet sich für die Modellierung vernetzter eingebetteter Systeme.(Schorcht et al., 2003; Liebezeit, Zerbe & Löffler, 2003; Salzwedel, 2004) Nachfolgend wird eine kurze Übersicht über das Toolsystem und die Modelierung gegeben, eine ausführlichere Auseinandersetzung mit dem Toolsystem MLDesigner findet sich beispielsweise in (Baumann, 2009). MLDesigner ist sowohl für einen Bottom-Up- als auch für einen Top-Down-Entwurfsprozess geeignet. Der Hauptaugenmerk liegt allerdings auf dem Top-Down-Design, bei dem ein Gesamtsystem iterativ verfeinert wird. Die Beschreibung basiert auf der Dokumentation (MLDesign Technologies, Inc., 2007).

Die Modellierung ganzer Systeme wird vereinfacht und eine Analyse und Verifikation von Architektur, Funktion und Leistung bereits in der Phase des Systementwurfs ermöglicht. Die Validierung beschränkt sich dabei nicht nur auf Missions- und Systemebene sondern ist prozessbegleitend zudem auf der Implementierungsebene realisierbar. Im Hinblick auf die Systemleistung können Auswirkungen von Entwurfs- und Implementierungsentscheidungen geprüft werden. Hierdurch wird die Gefahr von Entwurfsfehlern reduziert.

Innerhalb der Entwicklungsumgebung werden unterschiedliche Berechnungsmodelle unterstützt, diese werden Domänen genannt. Nachfolgende Domänen sind vorhanden.

- Discrete Event (DE) Domäne
- Dynamic Dataflow (DDF) Domäne
- Synchronous Dataflow (SDF) Domäne
- Boolean Dataflow (BDF) Domäne
- Continous Time/Discrete Event (CT/DE) Domäne
- Finite State Machine (FSM) Domäne
- Higher-Order Function (HOF) Domäne

2.3.5.3.2 Domänen Domänen in MLDesigner geben das Berechnungsmodell für das Modell und die Simulation eines Systems an. Für die Modellierung von vernetzten eingebetteten Systemen ohne detaillierte Berücksichtigung der physikalischen Eigenschaften eignen sich die DE-Domäne und die FSM-Domäne.

Discrete Event (DE) Domäne Die DE-Domäne basiert auf einem ereignisgesteuerten Berechnungsmodell. Der Auftrittszeitpunkt der Ereignisse innerhalb der Simulation ist frei wählbar. Partikel mit einem Zeitstempel repräsentieren die Ereignisse. In einer Ereigniswarteschlange werden die Ereignisse chronologisch

[16] `http://www.mldesigner.com` [08.01.2015]

nach den Zeitstempeln sortiert, die Verwaltung der Ausführung erfolgt über spezielle Abläufe.

Finite State Machine (FSM) Domäne Die FSM-Domäne ist keine eigenständige Domäne im eigentlichen Sinne, sondern ist eine Sub-Domäne, die in die System-Domäne eingebettet wird. Das Systemverhalten wird mit Zustandsautomaten beschrieben. Hier stehen Zustände und Transitionen zur Verfügung. Ein komplexes Verhalten kann beispielsweise mit Hierarchie, Ereignissen, Bedingungen und Aktionen beschrieben werden.

2.3.5.3.3 Modellierung in MLDesigner

Die Modelle zur Struktur- und Verhaltensbeschreibung eines Systems werden mit Hilfe von graphischen, hierarchisch strukturierten Blockdiagrammen erstellt.

In der Modellierungshierarchie bildet das System die oberste Ebene, es beinhaltet alle verwendeten Blöcke und besitzt keine direkten Inputs oder Outputs. Das System bildet die Simulationsebene. Ein Block selbst hat, wie in Abbildung 2.3.1 dargestellt, verschiedene Ausprägungen. Es kann ein elementarer Block sein (Primitive), welcher als FSM-Modell oder C/C++ Quellcode vorliegt, oder es handelt sich um einen hierarchischen Block, ein Modul, dessen interne Struktur sich wiederum aus Modulen und Primitiven zusammensetzt. Blöcke können verschieden Typen von Variablen beinhalten. Dies sind zum Beispiel Parameter, welche sich während der Simulation nicht ändern, oder Speicherelemente (Memory), welche im Gegensatz dazu veränderlich sind.

Variablen können Modullokal sein sowie mit einem gleichartigen Element in der übergeordneten Modellebene referenziert werden. Ein Datenaustausch zwischen Blöcken kann durch diese „verlinkten Variablen“ sowie durch Ein- und Ausgabe-Ports erfolgen. Ein Port hat einen festen Datentyp und einen Signalfluss. Durch Verwendung von Wormholes können Module verschiedener Domänen kommunizieren.

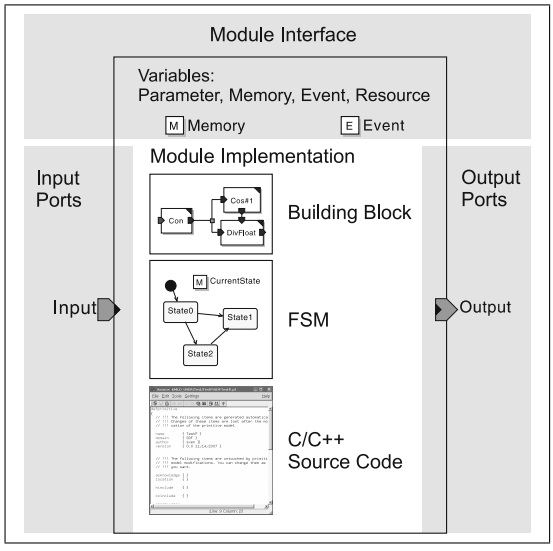


Abbildung 2.3.1: Module in MLDesigner

2.3.5.4 LabVIEW

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) von National Instruments[17] versteht sich nicht als grafische Programmiersprache sondern als Entwurfswerkzeug zum System Design (Graphical System Design) (Larsen, 2010; Georgi & Metin, 2009). Die Basis in LabVIEW bildet der synchrone Datenfluss, wobei ähnlich wie bei Matlab/Simulink ein Hauptaugenmerk auf einer plattform- bzw. hardwareunabhängigen Beschreibung liegt, die einen direkten Übergang zu einer Realisierung bildet (Platform Based Approach).

Ähnlich wie Matlab/Simulink hat LabVIEW seine Leistungsfähigkeit nach und nach erweitert. In (Jensen, Chang & Lee, 2011) erfolgt die Darstellung einer modellbasierten Design-Methode für Cyber-Physical-Systems mit LabVIEW und Ptolemy II.

In Bezug auf die Modellierung und Simulation von heterogenen Systemen mit unterschiedlichen Berechnungsmodellen ist somit auch LabVIEW, welches auf den ersten Blick zunächst nur als grafisches Programmiertool mit großer Leistungsfähigkeit im Bereich Rapid Prototyping anmutet, zu den Multi-Domänen Modellierungs- und Simulationstools zu zählen, welches zusätzlich Hierarchien unterstützt.

Die leistungsfähigen Berechnungsmodelle werden in (Jensen et al., 2011) aufgeführt. So erfolgt beispielsweise die Modellierung von kontinuierlichen Systemen im LabVIEW Control, Design, and Simulation Module mit Hilfe gewöhnlichen Differentialgleichungen und differenziell algebraischen Gleichungen. Innerhalb des LabVIEW Statechart Module findet sich eine Variante der Statecharts von Harel. Zudem steht natürlich der Datenfluss zur Verfügung.

Das zentrale Element in LabVIEW bilden die sogenannten Virtuellen Instrumente (VI), welche einen hierarchischen Aufbau haben können und Ein- und Ausgänge besitzen. Ein VI besteht aus den zwei Hauptkomponenten Frontpanel und Blockdiagramm. Das Frontpanel bildet mit Bedien- und Anzeigeelementen das Userinterface und definiert in Verbindung mit dem Connector-Block die Ein- und Ausgangssignale. Das Blockdiagramm enthält die funktionale Beschreibung eines VIs. Die Elemente eines Blockdiagramms sind selbst VIs, Funktionen in LabVIEW, Konstanten und Steuerungsstrukturen.

[17] `www.ni.com` [08.01.2015]

3 Entwicklung des Modellierungsansatzes und des zugehörigen Meta-Modells

3.1 Modellierung im Bereich Kommunikation

In (Zeigler et al., 2000) werden drei grundlegende Aufgabenstellungen im Themenbereich der Modellierung von Systemen identifiziert: *systems analysis*, *systems inference* und *systems design*. Unter *systems analysis* wird dabei verstanden, dass das Verhalten und die Eigenschaften des Systems untersucht werden sollen. Dabei spielt es keine Rolle ob das System bereits existiert oder ob es sich noch in der Planung befindet. Die beiden anderen Aufgabenstellungen unterscheiden sich insbesondere in diesem Punkt. Während bei der *systems inference* ein System bereits existiert und die Funktionalität von dem beobachtbaren Verhalten abgeleitet werden soll, ist im Bereich *systems design* das zu entwerfende System noch nicht in dem geplanten Zustand vorhanden. Ein geeignetes Design soll erst noch entwickelt werden.

Eine zentrale Herausforderung im Bereich der Modellierung ist stets die Komplexität und der Umfang der Modelle im Vergleich zu dem realen System. Die Größe und Komplexität eines Modells hängt dabei direkt mit dem Grad der gewählten Detaillierung zusammen.

In (Zeigler et al., 2000) werden einige Methoden zur Abstraktion aufgeführt, welche zur Reduzierung der Komplexität eines Modells verwendet werden können. Hierzu zählen beispielsweise die Aggregation (Zusammenfassen von mehreren Komponenten zu einer einzelnen Komponente, welche das gemeinsame Verhalten repräsentiert) und die Omission (nicht relevante Komponenten, Variablen oder Interaktionen werden ausgelassen). Zudem werden noch Linearisierung, Transformation von Formalismen und Tausch von stochastischen und deterministischen Beschreibungen identifiziert.

Bei der Verwendung von Abstraktionen gilt es zwischen den benötigten Ressourcen abzuwägen, insbesondere zwischen der Laufzeit von Simulationen und dem Verlust der Genauigkeit von Vorhersagen. Im Idealfall wird die Abstraktion methodisch unterstützt.

Bei der Untersuchung und Analyse von Systemen und deren Eigenschaften sind die Fragestellungen sowie die Anforderung an die Genauigkeit von Vorhersagen sehr unterschiedlich. In großen komplexen Systemen ist es zumeist nicht notwendig alle Untersuchungen mit einem detaillierten Modell durchzuführen, oftmals ist ein abstrakteres Modell, welches Komponenten und Funktionen kombiniert (Aggregation) oder nicht relevante Komponenten und Funktionen nicht berücksichtigt (Omission) ausreichend.

3.1.1 Ableitung von Abstraktionsklassen zur Modellierung der Kommunikation

Der zentrale Betrachtungspunkt dieser Arbeit ist die Modellierung von Bussystemen. Diese werden dabei als Teil eines Gesamtsystems betrachtet. Die Kommunikationskomponenten sind jeweils Bestandteil eines größeren Modells. Durch die explizite Berücksichtigung der Eigenschaften der Kommunikation können Auswirkungen dieser auf das Gesamtsystem berücksichtigt werden.

Die spezifische Ebene der Abstraktion und der Grad der Detaillierung sind im Allgemeinen sehr systemspezifisch. An dieser Stelle wird sich bei der Modellierung auf Kommunikationssysteme im Besonderen auf Bussysteme beschränkt. Den Schwerpunkt bilden als Beispiele FlexRay und CAN aus der Domäne Automotive.

Für den Datenaustausch von vernetzten Komponenten lassen sich unterschiedliche Grade der Detaillierung identifizieren, welche für eine Abstraktion bei der Modellierung von Kommunikation im Speziellen bei Nutzung von Bussystemen zum Tragen kommen. Nachfolgend werden vier grundsätzliche Abstraktionsklassen für die Modellierung definiert.

Die abstrakteste Sicht der Datenübertragung wird im Folgenden als *Level 0* bezeichnet. Hier wird in der Modellierung des Datenaustausches die Übertragungszeit auf Basis der Datenrate und der Größe der übertragenen Datenpakete bestimmt. Diese sehr einfache Annahme berücksichtigt keine spezifischen Eigenschaften der Übertragungsprotokolle. Ein Vergleich von Protokollen ist nur sehr eingeschränkt möglich, da zum einen weder der Protokolloverhead noch das Buszugriffsverfahren, welches eine Auswirkung auf den Ablauf und das zeitliche Verhalten der Kommunikation hat, berücksichtigt werden. Eine zusätzliche Berücksichtigung des Protokolloverheads verbessert die Aussagekraft unwesentlich und wird als Erweiterung der *Level 0* gesehen.

Die nachfolgenden Ebenen berücksichtigen zusätzlich das Buszugriffsverfahren des Kommunikationsprotokolls. In der nächsten Ebene (*Level 1*) erfolgt dieses durch die Einbeziehung von übergeordneten sehr einfachen Mechanismen des Buszugriffs. Diese werden auf einem sehr abstrakten Niveau mit Hilfe sehr einfacher Modelle integriert. Dies kann beispielsweise durch eine zentrale Zuteilung des Buszugriffs erfolgen.

In der folgenden Ebene *Level 2* wird nicht nur das Buszugriffsverfahren sondern auch die internen Protokollabläufe berücksichtigt. Hierzu zählen insbesondere die Mechanismen und Abläufe zur Steuerung des Buszugriffs. Diese Ebene kann sehr unterschiedliche Ausprägungen besitzen, da zum einen die Komplexität der Protokolle als auch der Detailgrad der Modellierung von einzelnen Abläufen, Mechanismen und Funktionen sehr stark variieren kann.

Die abschließende Ebene *Level 3* hat eine Sonderstellung. Sie berücksichtigt neben der reinen Kommunikation und den damit verbundenen Verzögerungszeiten zusätzlich die Inhalte der Kommunikation, d.h. die datenbezogene Verarbeitung auf Ebene der Anwendung oder des Gesamtsystems wird ermöglicht. Dies sind externe Aspekte des Kommunikationsprotokolls. Innerhalb der Kommunikation werden durch konkrete, anwendungsbezogene Daten Fehlererkennungs- und Fehlersicherungsmechanismen in die Simulation einbezogen. Eine Berücksichtigung von konkreten Daten ist unabhängig vom Detaillierungsgrad der Kommunikation. Darin besteht die Sonderstellung im Vergleich zu den vorherigen Ebenen.

3.1.2 Ableitung eines Ansatzes zur Modellierung der Kommunikation

Eine Erweiterung des klassischen Mission Level Design findet sich in (Baumann, 2009). Hier werden zunächst Defizite des allgemeinen Systementwurfs im Bereich der Betrachtung von Kopplungseffekten insbesondere in verteilten Systemen identifiziert. Ausgehend von dieser Betrachtung werden Methoden zur Bewertung von Systemarchitekturen und Unterstützung des Systementwurfs entwickelt. Eine standardisierte, ausführbare Architekturbeschreibung wurde entwickelt, welche ein flexibles Mapping von Funktion in Architektur und umgekehrt ermöglicht. Zusätzlich ist eine automatisierte Erzeugung von Gesamtsystemmodellen möglich.

Ein wesentlicher Bestandteil innerhalb dieses methodischen Ansatzes bezüglich Mapping von Funktion und Architektur ist die Berücksichtigung der Kopplung von Architekturkomponenten mit Hilfe von Kommunikationselementen, d.h. die Kommunikation ist Bestandteil der Untersuchung und das verwendete Kommunikationsprotokoll wird als Modellkomponente berücksichtigt. Die Modellierung der Kommunikationsprotokolle erfolgt in (Baumann, 2009) auf Basis des ISO/OSI-Referenzmodells und wird nachfolgenden als Referenz für eine ISO/OSI-orientierte Modellierung bezeichnet. Für die Kommunikation existiert zudem die Anforderung einer getrennten physikalischen Schicht.

Bei der Demonstration des Ansatzes werden Ethernet sowie TCP/IP-basierte Protokolle verwendet. Diese Protokolle haben zwei große Vorteile, sie basieren einerseits auf einer Punkt-zu-Punkt-Verbindung und andererseits existiert eine ISO/OSI-konforme Spezifikation der Protokolle. Diese Eigenschaften der verwendeten Protokolle harmonieren sehr gut mit den in (Baumann, 2009) präsentierten Methoden.

In der Praxis ist eine ISO/OSI orientierte Spezifikation von Protokollen nicht immer gegeben, was zumeist darauf zurückzuführen ist, dass insbesondere im Bereich der Feldbussysteme zumeist nur die unteren Ebenen des ISO/OSI-Referenzmodells adressiert werden. Diese ISO/OSI-orientierte-Modellierung ist insbesondere für die Anwendungsdomäne Automotive daher nicht optimal.

Durch die in (Baumann, 2009) verwendeten Punkt-zu-Punkt-Verbindungen ist die Modellierung der Kommunikation im Allgemeinen auf Bussysteme nicht übertragbar, dies ist insbesondere auf das zumeist verteilte Zugriffsverfahren zurückzuführen. Beispielsweise beim CAN-Bus liegt der Mechanismus zur Regelung des Buszugriffes innerhalb der einzelnen Knoten.

Auf Grund dieser Beobachtungen wird für diese Arbeit ein anderer Ansatz verfolgt, welcher die spezifischen Eigenschaften der primären Anwendungsdomäne und der Referenzprotokolle FlexRay und CAN berücksichtigt. Ziel ist dabei eine anwendungsbezogene und Hardware-Komponenten orientierte Modellierung welche eine flexible Kopplung ermöglicht. Unter der flexiblen Kopplung werden dabei verschiedene Aspekte zusammengefasst: Verwendung unterschiedlicher Abstraktionsebenen, heterogene Kommunikation im Sinne einer Kopplung unterschiedlicher Protokolle, Erweiterbarkeit und Integration sowie Interaktion mit anderen Standards (beispielsweise AUTOSAR). Das nachfolgend vorgestellte Konzept ermöglicht eine variable Adaption und die Integration der Kommunikationsmodelle in andere Standards und Konzepte.

3.2 Darstellung des Ansatzes zur Modellierung verteilter eingebetteter Systeme

Das in dieser Arbeit entwickelte Modellierungskonzept folgt einem Dienst- und Schichten-orientierten Aufbau, der sich an dem strukturellen Aufbau der Protokolle und der Hardware- und Funktionsarchitektur der (Steuer-)Geräte anlehnt.

Einschränkungen an die Struktur der Architektur durch Standardisierungen wie beispielsweise AUTOSAR, welcher sich mit der Festlegung einer einheitlichen Struktur für Elektrik/Elektronik-Architekturen in Kraftfahrzeugen befasst, ergeben sich nicht. Das entwickelte Konzept eignet sich zur Kombination und Integration mit konkreten Systemmodellen welche einem Schichten-orientierten Aufbau folgen.

Im ersten Schritt wird eine grundlegende Modell-Architektur für eine homogene Kommunikation abgeleitet und diese über ein Meta-Modell präzisiert. Anschließend erfolgt mit Hilfe einer Erweiterung die Berücksichtigung komplexer Systemarchitekturen, welche sich durch eine heterogene Kommunikationsinfrastruktur auszeichnen.

3.2.1 Single Communication System (SCS)

Die grundlegende Architektur (Basis-Architektur) der Modelle gliedert sich in drei Bereiche und wurde in (Klöckner, Köhler & Fengler, 2008) bereits vorgestellt. Der prinzipielle Aufbau wird in der Abbildung 3.2.1 verdeutlicht. Die dargestellten drei Schichten bilden die obere Ebene der Modellstruktur eines einzelnen Knotens und werden im Weiteren als Host, Communication Controller (CC) und Channel bezeichnet.

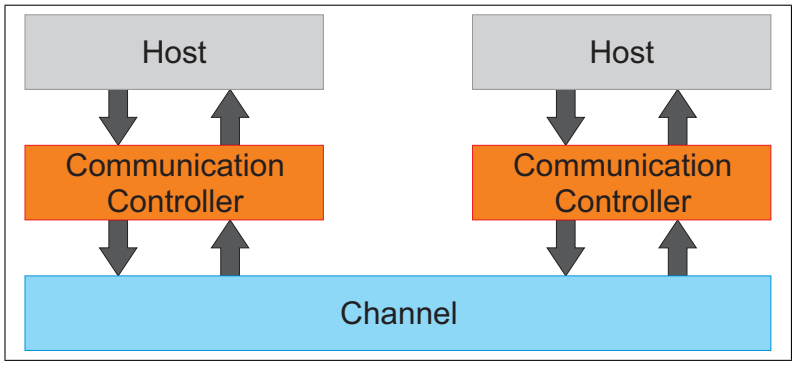


Abbildung 3.2.1: Basis Architektur für die Modellierung

Der Verbund eines Host-Elementes und eines CC-Elementes wird Knoten (Node) genannt, dieser beinhaltet somit die Anwendung und die Kommunikation. Innerhalb des CC-Elementes ist das Kommunikationsprotokoll mit den wichtigen Funktionen zum Datenaustausch wie beispielsweise Synchronisation, Fehlererkennung und Nachrichtenmanagement realisiert. Über durch den CC angebotene Dienste/Funktionen kann die Anwendung (Host) Kommunikationsmechanismen verwenden.

Die nutzerspezifische Anwendung ist innerhalb des Host-Modells beschrieben. Zudem enthält dieses Element einen spezifischen Funktionsblock, welcher den protokoll-

spezifischen Zugriff auf den CC realisiert und so die Kommunikation konfiguriert, Daten sendet und empfängt.

Die physikalischen Eigenschaften werden innerhalb des Channel-Elementes modelliert. Dieses kann auf sehr unterschiedlichen Abstraktionsebenen geschehen. So können die physikalischen Eigenschaften sehr detailliert modelliert werden oder lediglich das zeitliche Delay in die Analyse des Gesamtsystems einfließen. Zusätzlich können unterschiedliche Modelle zur Einbringung von Übertragungsfehlern integriert werden.

In den unterschiedlichen Modellen zu den betrachteten Bussystemen (siehe Kapitel 4) lassen sich diese Elemente wiederfinden. Eine ergänzende hierarchische Unterstrukturierung der einzelnen Schichten ist möglich und auch vorgesehen. Eine Erweiterung der Basis-Architektur wird in den Abschnitten zur Verbindung unterschiedlicher Bussysteme über Gateways und der Modellierung auf Applikationsebene unter Einbeziehung von Betriebssystemfunktionalitäten angewendet (Kapitel 3.2.4).

Die schichten- und dienst-orientierte Modellierung ermöglicht durch die modularisierte Verwendung der einzelnen Elemente und die Festlegung von Schnittstellen eine Integration in komplexe Strukturen bzw. Architekturkonzepte.

Die einzelnen Schichten der Basis-Architektur werden nachfolgend in der Reihenfolge von unten nach oben näher betrachtet.

3.2.1.1 Channel

Die Verknüpfung der einzelnen Communication Controller (CC) der Kommunikationsteilnehmer wird über das Channel-Element realisiert. Über dieses Element wird die Kommunikationsstruktur des modellierten Gesamtsystems festgelegt.

Der Channel empfängt Daten von den angeschlossenen Teilnehmern und leitet diese an alle Teilnehmer weiter, somit übernimmt er die Aufgabe der physikalischen Verbindung. Die Komplexität kann dabei abhängig vom Grade der Abstraktion von einzelnen Bits oder Signalpegeln bis hin zu kompletten Datenframes reichen. Abhängig vom Detailgrad der Modelle können dem Channel u.a. folgende Funktionen zugeordnet werden: Zeitverzögerung der Datenübertragung, Einstreuung von Fehlern und Berechnung des Buszustandes.

Im Rahmen der Arbeit liegt der Schwerpunkt der Modelle der Bussysteme nicht auf der detaillierten Modellierung der physikalischen Eigenschaften. Aufgrund dieser Einschränkung wird eine Abstraktion von der physikalischen Übertragung vorgenommen. Im Allgemeinen wird in den Modellen innerhalb der Arbeit von der Kommunikation mittels Datenstrukturen unterschiedlicher Komplexität ausgegangen. Eine Anpassung der Modelle im Sinne einer Erweiterung oder Integration von Modellen zur detaillierten Einbeziehung physikalischer Effekte im Rahmen einer physikalischen Übertragung der Daten, basierend auf Spannungspegeln, ist durch Substitution und hierarchische Verfeinerung eines Channel-Elementes problemlos möglich.

3.2.1.2 Communication Controller

Während der Channel das Übertragungsmedium und ggf. noch die Bitübertragungsschicht repräsentiert, umfasst der Communication Controller die wesentlichen Funktionen des Kommunikationsprotokolls.

Im Falle der beiden Bussysteme FlexRay und CAN decken diese Funktionen der beiden unteren Schichten des ISO/OSI-Referenzmodells ab. In diesem Fall kann daher der CC als Modell dieser beiden Schichten angesehen werden. Im Allgemeinen wird keine Unterstrukturierung des CC in ISO/OSI-Schichten vorgenommen. Das vorgestellte Konzept nimmt hier allerdings auch keine Einschränkung vor.

Im Hinblick auf eine Modellierung und Simulation auf unterschiedlichen Abstraktionsniveaus, werden auf der Ebene des CC unterschiedliche Abstraktionsstufen eingeführt. Dieses wird in Abschnitt 4.1.3.1 am Beispiel des FlexRay demonstriert.

Die Hauptaufgaben des CC liegen im Senden und Empfangen von Nachrichten sowie in der Steuerung des Medienzugriffs. Diese und weitere Funktionalitäten werden in Form von Diensten der übergeordneten Schicht, dem Host, zur Verfügung gestellt. Neben diesen Basis-Funktionen müssen weiterhin Dienste zur Verfügung gestellt werden, die eine Konfiguration und Einstellung von Kommunikationseigenschaften für eine servicebasierte Steuerung durch den Host erlauben. Der konkrete Umfang der Dienste ist abhängig von dem jeweiligen Protokoll.

Die Dienste selbst und auch der Zugriff auf diese müssen für alle Varianten eines Protokolls, welche sich durch ihr Abstraktionsniveau unterscheiden, identisch sein. Die einheitliche Schnittstelle erlaubt so einen einfachen Austausch und einen hohen Grad an Wiederverwendbarkeit im Hinblick auf Modelle und Simulationen auf der Ebene des Gesamtsystems.

3.2.1.3 Host

Das abschließende Element in der Basis-Architektur bildet der Host. In diesem Bereich findet sich die Applikationsebene wieder. Diese, dem Kommunikationsprotokoll übergeordnete Schicht, ist ein Stellvertreter oder Platzhalter-Element und kann unterschiedliche Funktionalitäten, Realisierungen und Ausprägungen repräsentieren.

Die konkrete Ausprägung ist abhängig von den Anforderungen des Gesamtsystems. Das Host-Element kann ein einfacher Layer sein, welcher die Kommunikation lediglich um eine sehr einfache Funktionalität ergänzt oder auch eine komplexe Applikation mit mehreren funktionalen hierarchischen Schichten.

Die konkrete Funktionalität kann daher variieren. Unter dem Begriff der überliegenden Schicht ist somit die Gesamtheit der übergeordneten Schichten zu verstehen bzw. Funktionen die der Gesamtheit zugeschrieben werden und nicht Bestandteil des Kommunikationsprotokolls sind.

Im Falle von FlexRay, welches innerhalb des ISO/OSI-Schichtenmodells die beiden unteren Schichten den Physical Layer und den Data Link Layer abdeckt, können dies zum Beispiel ein übergeordnetes Transportprotokoll sein, eine API oder direkt eine Anwendung die Daten sendet und empfängt. Der Host wird als abstraktes Gebilde gesehen, dessen Struktur nur eingeschränkt bekannt ist und auf die vom Communication Controllers angebotenen Services nutzt. Diese Services bilden die Schnittstelle zwischen CC und Host.

3.2.2 Abbildung der SCS-Modellarchitektur auf die Kommunikationsprotokolle im Segment Automotive

Diese grundlegende Modell-Architektur wird im nächsten Schritt auf die Kommunikationssysteme im Automobil abgebildet. Dies soll die Eignung der gewählten Modellpartitionierung für die Domäne demonstrieren. Neben den beiden zentralen Kommunikationsprotokollen FlexRay und CAN werden an dieser Stelle ergänzend auch der LIN-Bus (siehe Anhang A.2.1) sowie der MOST (Media Oriented Systems Transport) (Grzemba, 2007) betrachtet. Zur Demonstration der prinzipiellen Anwendung wird die entwickelte Modellarchitektur auf die automotive Kommunikation abgebildet.

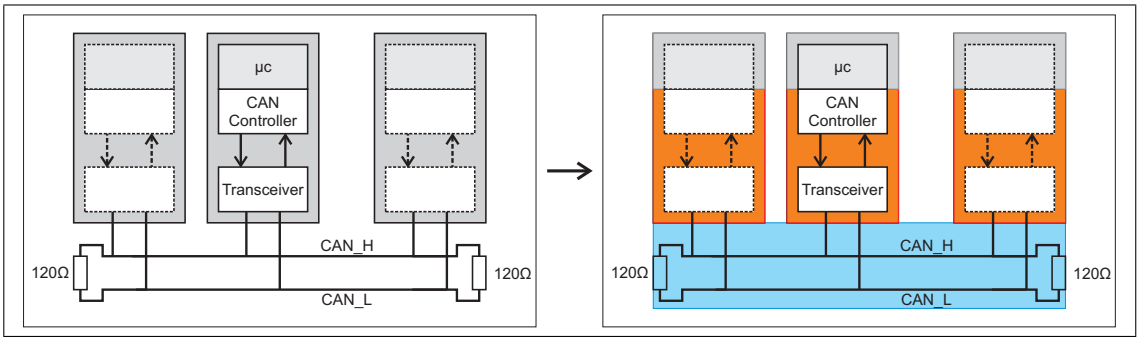


Abbildung 3.2.2: Abbildung der Struktur des CAN-Bus in die entwickelte Modell-Architektur (Zimmermann & Schmidgall, 2011)

In Abbildung 3.2.2 ist auf der linken Seite eine Struktur des CAN-Bus dargestellt welche (Zimmermann & Schmidgall, 2011) entnommen wurde. Hier findet sich ein Element Mikrocontroller (μC) für die Realisierung der anwendungsspezifischen Funktionen, ein Element CAN Controller für die Steuerung der Kommunikation, ein Transceiver und ein Kommunikationskanal. Auf der rechten Seite der Grafik findet man die Darstellung des CAN welche entsprechend der entwickelten Modellarchitektur eingefärbt wurde. Man erkennt anhand der farblichen Kennzeichnung die Elemente **Host** (grau), **Communication Controller** (orange) und **Channel** (blau). Damit zeigt sich die Eignung der Modellarchitektur für die Modellierung eines Systems, welches den CAN-Bus zur Kommunikation verwendet.

Neben dem CAN ist der FlexRay das zentrale Beispiel für die Modellierung. In Abbildung 3.2.3 zeigt sich analog auf der linken Seite die Struktur eines FlexRay-Knotens bestehend aus den Elementen Host Controller, FlexRay Controller, zwei Treiberbausteinen für den Bus und die beiden Kommunikationskanäle. Auf der rechten Seite der Abbildung wird die Struktur des FlexRay-Knotens in die Modellarchitektur abgebildet.

In einem abschließenden Schritt werden ergänzend der LIN und der MOST betrachtet. In Abbildung 3.2.4a ist oben die Struktur eines LIN dargestellt. Diese Struktur basiert auf der Darstellung in (Grzemba & Wense, 2005; LIN Consortium, 2010) und wurde um einen Funktionsblock für anwendungsspezifische Funktionen ergänzt. Man

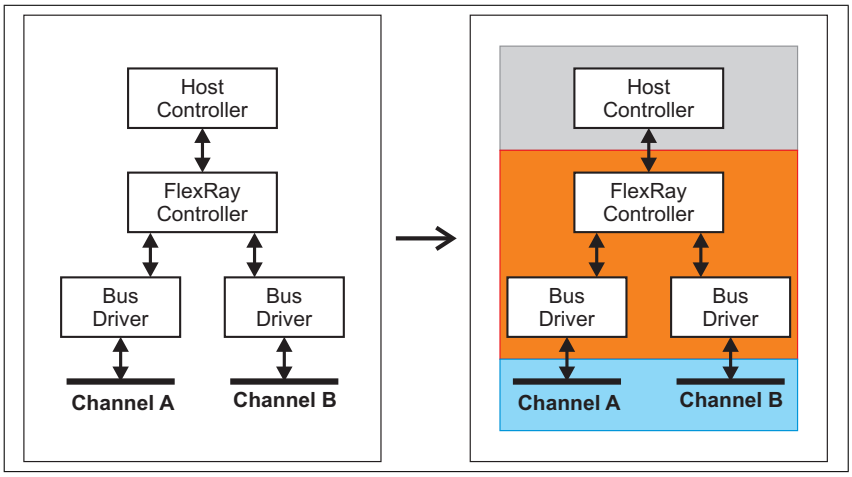
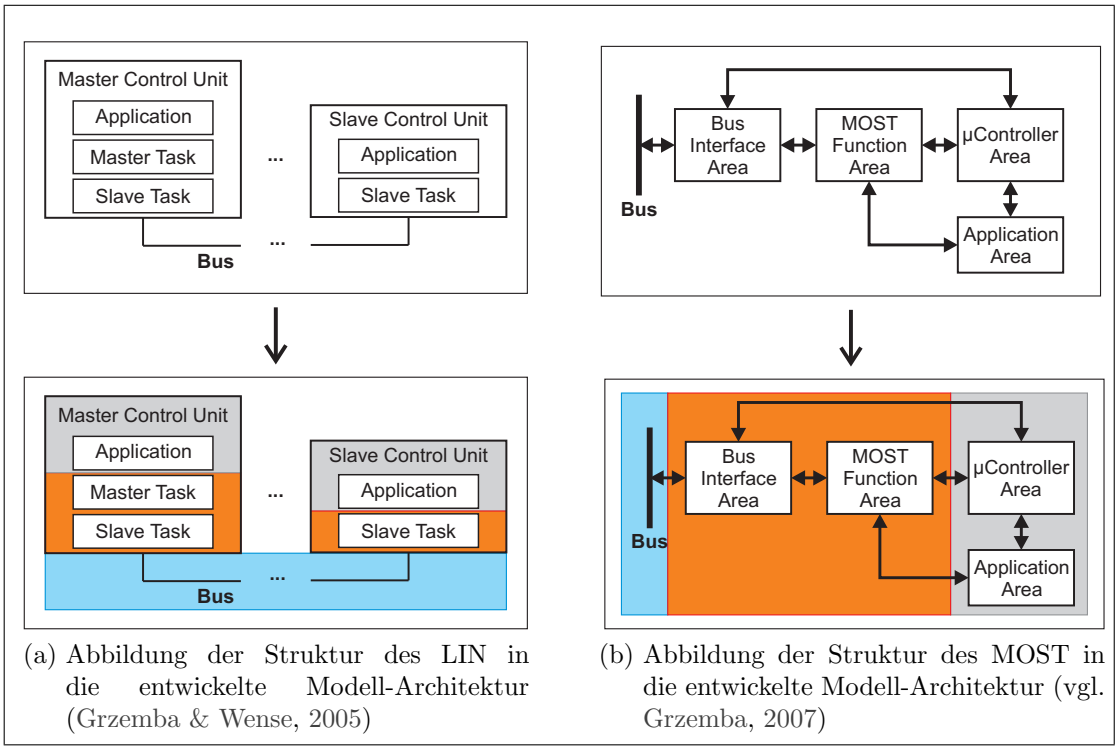


Abbildung 3.2.3: Abbildung der Struktur des FlexRay in die entwickelte Modell-Architektur (Rausch, 2008)



(a) Abbildung der Struktur des LIN in die entwickelte Modell-Architektur (Grzemba & Wense, 2005)

(b) Abbildung der Struktur des MOST in die entwickelte Modell-Architektur (vgl. Grzemba, 2007)

Abbildung 3.2.4: Abbildung der Struktur des LIN und des MOST in die entwickelte Modell-Architektur

erkennt zwei über den Bus verbundene Kommunikationsteilnehmer, einen Master- und einen Slave-Knoten. In den einzelnen Knoten finden sich die LIN-spezifischen Funktionsblöcke Anwendung, Master Task (Steuerung der Kommunikation) und Slave Taks (Senden von Daten). In der unteren Abbildung wurden die einzelnen Bestandteile der Modellarchitektur wie zuvor über eine farbliche Kennzeichnung zugeordnet.

Zuletzt erfolgt die Abbildung der Struktur eines MOST-Knotens in die Modellarchitektur. In Abbildung 3.2.4b ist eine vereinfachte Darstellung eines MOST-Knotens

erkennbar (vgl. Grzemba, 2007). Ein MOST-Knoten setzt sich aus einem anwendungs- bzw. gerätespezifischen Teil und einem kommunikationsspezifischen Teil zusammen. Der Anwendungsteil besteht aus der μController Area und der Application Area und wird in der Modellarchitektur dem Element Host zugeordnet. Der Kommunikationsteil beinhaltet die MOST Function Area sowie die Bus Interface Area und wird auf das Modellelement Communication Controller abgebildet. Als letztes erkennt man noch den Bus welcher die Verbindung der einzelnen Knoten herstellt.

Die klassischen Kommunikationssysteme im Automobilbereich (FlexRay, LIN, CAN und MOST) können mit Hilfe bzw. basierend auf der definierten grundlegenden Modellarchitektur modelliert werden. Um den Ansatz möglichst offen und flexibel zu halten, werden an dieser Stelle keine weiteren Spezialisierungen vorgenommen, da diese zu Einschränkungen beim Transfer in andere Anwendungsdomänen führen könnten.

3.2.3 Ableitung des SCS Meta-Modells

Die einzelnen Modellbestandteile werden durch die Ableitung eines Meta-Modells aus der grundlegenden Architektur etwas detaillierter betrachtet. Anhand des Meta-Modells wird im Kapitel zu der konkreten Modellierung der Kommunikationsprotokolle CAN und FlexRay eine Einordnung der entwickelten Modellkomponenten vorgenommen.

Bisher wird bei der Modellierung von einem Single Communication System ausgegangen, d.h. in einem System existiert genau ein Kommunikationssystem (Bus) über den alle Teilnehmer miteinander verbunden sind. Mit Beachtung dieser Eigenschaft erfolgte die Ableitung des zugehörigen Meta-Modells, diese ist in Abbildung 3.2.5 dargestellt.

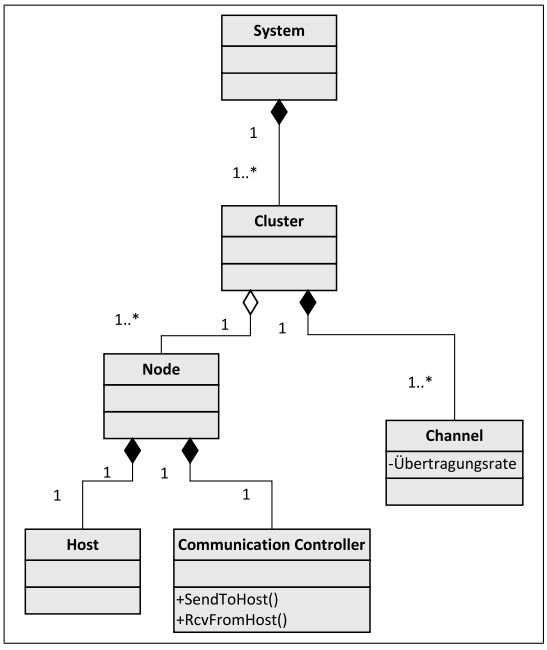


Abbildung 3.2.5: Meta-Modell für das Single Communication System

Das oberste Element bildet das System, welches in einem Single Communication System aus genau einem Cluster besteht. Der Cluster repräsentiert das gesamte Netzwerk, alle Knoten im System die über ein Kommunikationsmedium miteinander verbunden sind. Das Kommunikationsmedium ist der Channel, welcher die Knoten (Node) zu einem Kommunikationsverbund zusammenschließt. Zu einem Cluster können dabei ein oder mehrere Channel zugeordnet sein, abhängig davon ob es sich um eine ein- oder mehrkanalige Topologie handelt. Im Falle des FlexRay werden bis zu zwei Übertragungskanäle unterstützt.

Einem Cluster werden über den Channel ein oder mehrere Kommunikationsteilnehmer (Node) zugeordnet. Ein einzelner Knoten setzt sich dabei analog zur bereits vorgestellten Basisarchitektur aus einem Host und einem Communication Controller zusammen. Der Communication Controller übernimmt dabei die Aufgaben der Kommunikation, d.h. die Datenübertragung zwischen einzelnen Knoten genauer gesagt zwischen zwei Host über den Channel. Wesentlicher Bestandteil des Communication Controller sind die Funktionen zum Senden und Empfangen von Daten, welche dem Host als Dienste angeboten werden (*SendToHost()*, *RcvFromHost()*). Der Host selbst repräsentiert alle weiteren nicht kommunikationsspezifischen Funktionen. Hierzu zählen insbesondere die anwendungsspezifischen Funktionen, dies sind beispielsweise die Signal- und Datenverarbeitung sowie die konkrete Steuerung und Regelung.

3.2.4 Heterogeneous Communication System (HCS)

Für die Modellierung komplexer Systeme ist diese einfache Basisstruktur nicht ausreichend. Innerhalb einer erweiterten Architektur, werden zwei zusätzliche Aspekte berücksichtigt, um zusätzliche und weiterführende Interoperabilitäten zu untersuchen. Diese erweiterte Systemarchitektur welche in erster Linie eine heterogene Kommunikation berücksichtigt wurde in (Klöckner, Müller, Fengler & Huang, 2009) bereits vorgestellt. Betrachtet man insbesondere Systeme im Automobilbereich, so wird deutlich, dass die systemweite Kommunikation nicht homogen ist. Innerhalb eines einzelnen Systems, kommen eine Vielzahl von unterschiedlichen Kommunikationssystemen zum Einsatz. Teilnehmer tauschen Daten untereinander aus und beschränken sich dabei nicht auf ihr eigenes Subsystem. Dieser Nachrichtentransfer über unterschiedliche Netze mit unterschiedlichen Eigenschaften wie Zugriffsverfahren, Zeitverhalten, Datenrate und ähnlichem beeinflusst erheblich das Verhalten des Gesamtsystems. Die Kopplung von Bussystemen erfolgt mit Hilfe von Gateways.

Eine Anpassung des Basismodells erfolgt nachfolgend auf Applikationsebene, d.h. die Schicht Host wird weiter aufgeteilt. In Abbildung 3.2.6 wird die prinzipielle Strukturanpassung dargestellt. Die Zusatzelemente sind zum einen ein Gateway (GW), welches innerhalb des linken Netzwerk-Knoten eingeordnet ist, und zum anderen Betriebssystemfunktionen (OS), welche innerhalb des rechten Netzwerk-Knoten dargestellt sind. Beide Erweiterungen werden der Applikationsebene zugeordnet und die unterliegenden Schichten (Communication Controller und Channel) werden hierdurch nicht beeinflusst.

Am konkreten Beispiel der Erweiterung zur Berücksichtigung heterogener Netze

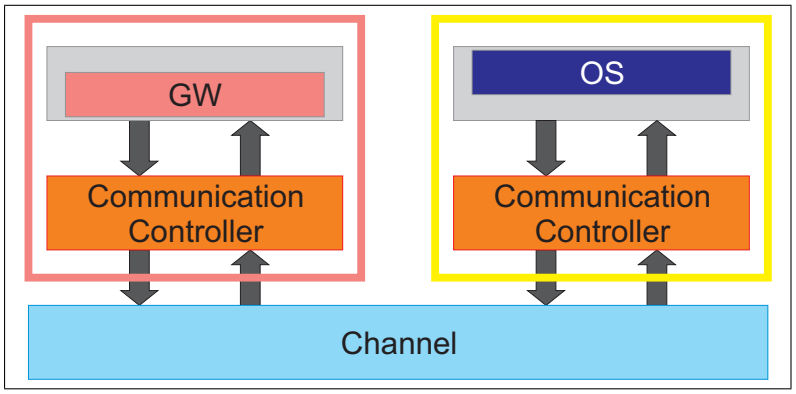


Abbildung 3.2.6: Erweiterte Architektur für die Modellierung

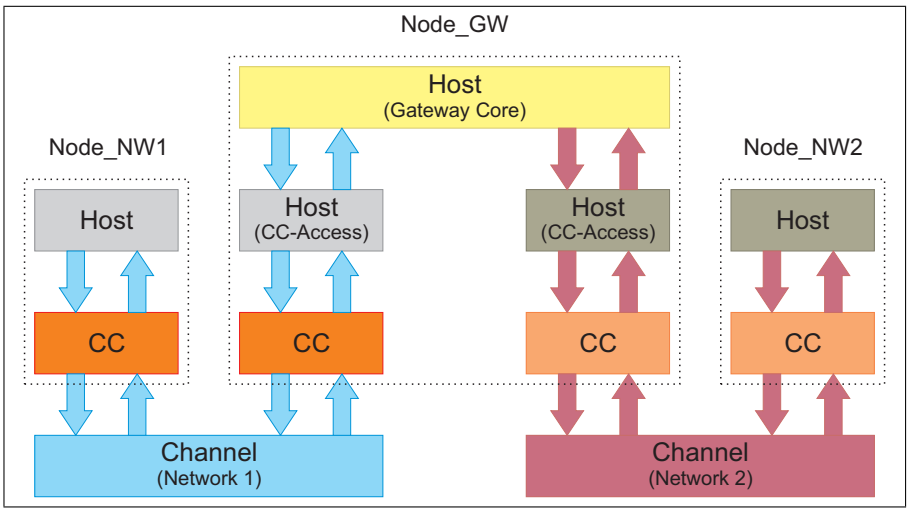


Abbildung 3.2.7: Gateway Architektur für die Modellierung

wird die so erweiterte Basisarchitektur erläutert.

Innerhalb der Basis-Architektur ist ein Netzwerk-Knoten eine Kombination aus Host und Communication Controller. In Abbildung 3.2.7 entsprechen die beiden Knoten (Node_NW1 und Node_NW2), die nur eine Netzwerkanbindung besitzen, dieser Definition. Der zwischen den Netzen vermittelnde Knoten (Node_GW) ist Teilnehmer der beiden anderen Netze. Dieser spezielle Knoten hat folgende Eigenschaften und Komponenten. Der Knoten ist verbunden mit zwei unterschiedlichen Netzen. Für diese Anbindung besitzt er Zugangspunkte zu zwei unterschiedlichen Channeln. Der Zugriff auf diese einzelnen Channel erfolgt über zugeordnete Communication Controller. Die Channel und CCs haben dabei die identischen Aufgaben und auch den identischen Aufbau und Struktur wie innerhalb der Basis-Architektur. Der Host wird nun in zwei Ebenen und drei Komponenten aufgeteilt. Die untere Ebene beinhaltet zwei Komponenten (Host CC-Access), diese realisieren den Zugriff auf das jeweilige Protokoll (Communication Controller). Die übergeordnete Ebene enthält eine Komponente (Gateway-Core), welche einen speziellen Applikationsteil enthält und mit den Kernfunktionen des Gateway so die Vermittlung zwischen den beiden Netzen realisiert.

Ein spezieller Knoten, welcher die Funktionalität eines Gateways realisiert, ist somit eine Kombination von unterschiedlichen Host-Subkomponenten zum Zugriff auf die spezifischen Netze sowie einem Gateway Core und den zugeordneten Communication Controllern. Dieses Gateway-Konzept integriert sich nahtlos in das grundlegende Modellierungskonzept und folgt den Anforderungen an ein Gateway. Ein konkretes Modell für ein Gateway wird in Abschnitt 4.2.2 vorgestellt.

Der nächste Schritt zur Erhöhung des Detaillierungsgrades der Modelle erfolgt durch die Annahme, dass innerhalb eines Knotens getrennte Applikationen existieren, welche in einzelne Tasks aufgeteilt werden können. Die Erweiterung der Architektur ermöglicht die Berücksichtigung weiterer Einflussfaktoren auf das Systemverhalten, wie beispielsweise das Scheduling von Tasks. Ein konkretes Modell für ein Betriebssystem wird in Abschnitt 4.3 vorgestellt.

3.2.5 Übergang vom SCS zum HCS Meta-Modell

Die Erweiterung der Basisarchitektur wird nachfolgend auch innerhalb des Meta-Modells durchgeführt. Das Resultat ist in Abbildung 3.2.8 zu sehen.

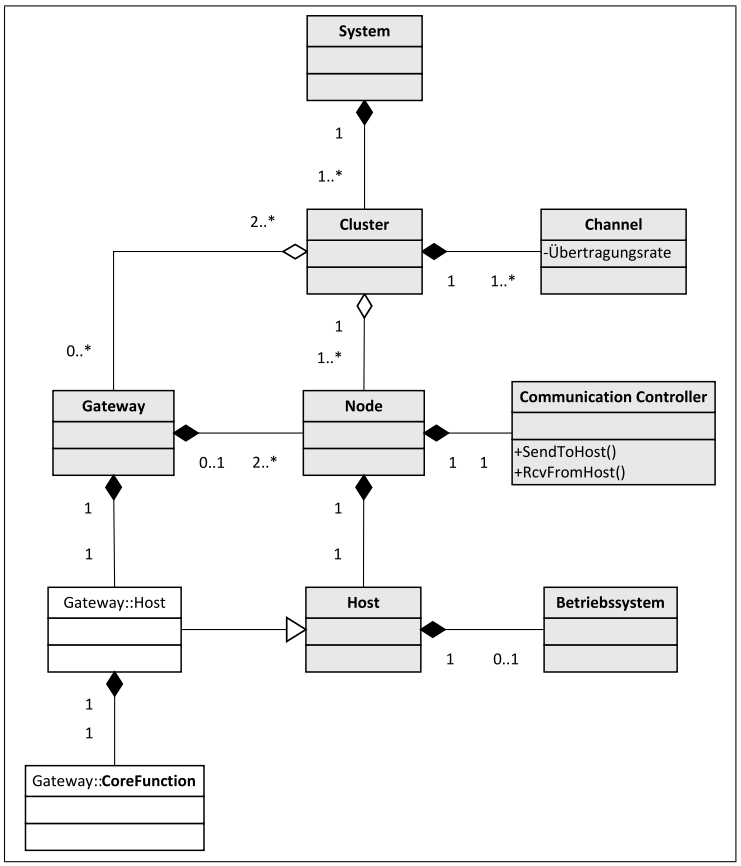


Abbildung 3.2.8: Meta-Modell für das Heterogeneous Communication System

Die Elemente System, Cluster, Channel, Node, Host und Communication Controller finden sich aus dem SCS-Meta-Modell wieder. Hinzugekommen sind die Elemente

Gateway und Betriebssystem, welche an dieser Stelle ein wenig verfeinert werden. Die Basiselemente des Meta-Modells sind gefärbt, um diese in den späteren Abschnitten jeweils als Basiselemente des Meta-Modells zu identifizieren. Ein System besteht nun aus einem oder mehreren Cluster, in einem System können sich nun mehrere Netze bzw. Subnetze befinden. Die Eigenschaften von Cluster, Channel, Node, Host und Communication Controller sind analog zu dem zuvor betrachteten SCS-Meta-Modell.

Das neue Element Gateway ist ein Teil von mindestens zwei Clustern und bildet somit einen Verbindungspunkt zwischen einzelnen Clustern. In einem einzelnen Cluster kann eine beliebige Anzahl von Gateways existieren. Ein Gateway selbst besteht aus mehreren Knoten (Node), welche jeweils den Zugangspunkt zu einem Cluster darstellen. Die Vermittlung zwischen den an einem Gateway angeschlossenen Clustern erfolgt über einen spezifischen Gateway-Host, welcher wiederum zusätzlich zu seinen Basisfunktionen zur Weiterleitung von Daten beliebige weitere anwendungsspezifische Funktionen beinhalten kann.

Ein weiteres zusätzliches Element bildet das Betriebssystem. Dieses ist ein optionaler Bestandteil eines Host. Als Verfeinerung können hier die Betriebssystemstandards OSEK-OS und OSEKtime-OS eingeordnet werden.

3.2.6 Abbildung der HCS-Modellarchitektur auf die Kommunikationsstruktur im Segment Automotive

Nach der Darstellung des HCS-Meta-Modells soll an dieser Stelle auch eine strukturelle Abbildung erfolgen. Während für die Struktur des Single Communication System die jeweiligen Kommunikationsprotokolle als Referenz dienen, reicht ein einzelnes Protokoll für diese Abbildung im heterogenen Umfeld nicht aus. Zu diesem Zweck wird als Ziel der Abbildung der Beschreibungsstandard für Netzwerke FIBEX ausgewählt. Mit diesem Standard lassen sich die Kommunikationssysteme in einem Automobil und somit heterogene Architekturen definieren.

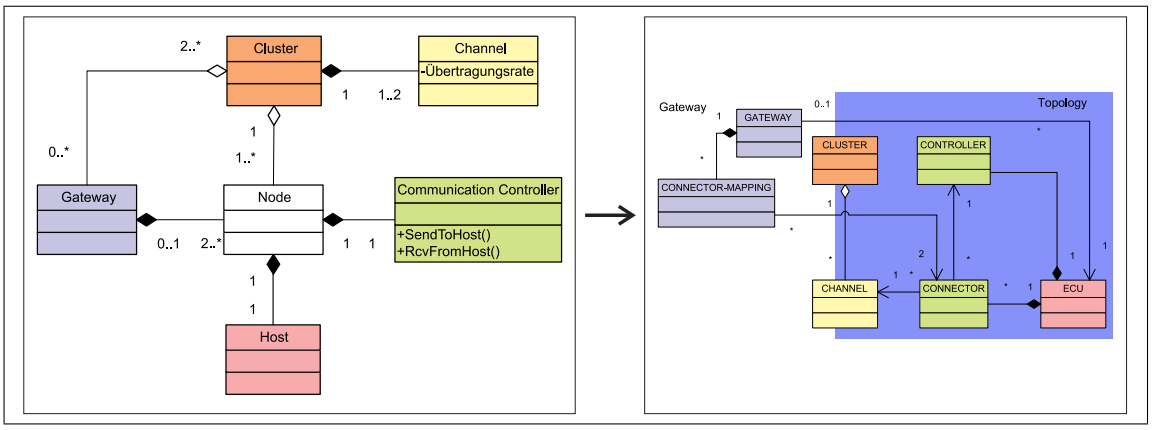


Abbildung 3.2.9: Abbildung eines Auszugs der FIBEX-Struktur auf das HCS-Meta-Modell

Zur Abbildung wird jeweils ein Ausschnitt aus dem HCS-Meta-Modell (vgl. Abbildung 3.2.8) sowie aus der UML-Repräsentation des FIBEX-Standards ausgewählt

(vgl. Abbildung 2.1.12). Die beiden Ausschnitte sind in der Abbildung 3.2.9 dargestellt. Zwischen den einzelnen Komponenten erfolgt eine farbliche Zuordnung.

Der Auszug aus dem HCS-Meta-Modell ist auf der linken Seite der Grafik dargestellt und besteht aus den Elementen Cluster, Channel, Node, Gateway, Host und Communication Controller. Auf der rechten Seite der Abbildung findet sich die FIBEX-Struktur mit den ausgewählten Bereichen Gateway und Topology (basierend auf Abbildung 2.1.12) und den Elementen, Cluster, Channel, Controller, Connector, Gateway, Connector-Mapping und ECU. Über die farbliche Codierung erfolgt die Visualisierung der Zuordnung zwischen dem HCS-Meta-Modell und FIBEX.

4 Modellierung der Systemkomponenten unter Anwendung des definierten Meta-Modells

4.1 Modelle ausgewählter Kommunikationsprotokolle unter Berücksichtigung der Abstraktionsebenen

In Abschnitt 3.1.1 wurden vier Abstraktionsklassen für die Modellierung der Kommunikation identifiziert, dabei adressieren die beiden Ebenen *Level 1* und *Level 2* jeweils das protokollspezifische Zugriffsverfahren. Modelle für diese beiden Abstraktionsklassen werden im Weiteren für die Demonstration des im vorherigen Abschnitt vorgestellten Modellierungskonzeptes verwendet.

Zunächst erfolgt zur Vorbereitung der Modellierung eine allgemeine Klassifikation von Buszugriffsverfahren, hier werden die Zugriffsverfahren der Protokolle FlexRay, CAN und TTP betrachtet. Diese Klassifikation ist der Ausgangspunkt für die Modelle auf der Abstraktionsebene *Level 1*, in der die abstrakten Buszugriffsverfahren betrachtet werden. Für die beiden zeitgesteuerten Protokolle FlexRay und TTP/C werden anschließend Modelle für das abstrakte Zugriffsverfahren vorgestellt. Ergänzend werden zum Vergleich an wenigen Stellen Modelle anderer Autoren betrachtet. Die Berücksichtigung und Darstellung des TTP/C, der in der Domäne Automotive eine untergeordnete Rolle spielt, hat einen vergleichenden Charakter. Auf Basis der abstrakten Zugriffsverfahren werden unterschiedliche Modellierungstechniken verwendet, so kommen Petri-Netze, Finite-State-Machines und das Multi-Domänen-Tool MLDesigner zum Einsatz.

Für die Modelle der Ebene *Level 2* kommt anschließend zur Modellierung ausschließlich das Multi-Domänen-Tool MLDesigner zum Einsatz. Betrachtet werden die Protokolle FlexRay und CAN. Im Falle von FlexRay werden zusätzlich unterschiedliche Verfeinerungen innerhalb der Abstraktionsebene berücksichtigt.

4.1.1 Allgemeine Klassifikation der Zugriffsverfahren

Die Klassifikation und Einteilung von Bussystemen (allgemeiner von Kommunikationssystemen) kann auf unterschiedliche Art und Weise erfolgen. Nachfolgend erfolgt die Klassifikation anhand des Buszugriffsverfahrens (Regel für das Senden von Nachrichten), da dem Buszugriffsverfahren im Rahmen der Arbeit ein besonderer Stellenwert eingeräumt wird. (Zimmermann & Schmidgall, 2011; Wallentowitz & Reif, 2006)

Auf der ersten Ebene können zwei Typen von Buszugriffen identifiziert werden: deterministische und stochastische Buszugriffe. Für die erste Gruppe der Zugriffsverfahren mit einem deterministischen Buszugriff gilt, dass ein Verfahren oder Algorithmus existiert, welcher den Zugriff auf das geteilte Medium steuert. Die einzelnen Antwortzeiten sind hierdurch vorhersagbar. Zu dieser Gruppe zählen unter anderem Master-Slave-Verfahren (Beispiel LIN), Token-Passing-Verfahren (Beispiel: Profibus) und TDMA-Verfahren (Beispiel: FlexRay und TTP/C).

Master-Slave-Verfahren Bei dem Master-Slave-Verfahren gibt es einen übergeordneten Busteilnehmer, der die Steuerung der Kommunikation übernimmt. Dies erfolgt meist in der Kombination eines Mastertelegramms (oder Headernachricht), mit dieser spricht der Master einen Slave an, und einem Slavetelegramm (oder Antwortnachricht) mit dem der Slave die angeforderten Daten sendet. Von großem Vorteil ist die einfache Organisation.

Token-Passing-Verfahren Beim Token-Passing-Verfahren wird das Senderecht (Token) von einem Teilnehmer zum nächsten Übergeben. Durch eine Beschränkung der maximalen Sendezeit (Token-Haltezeit) wird der Determinismus des Verfahrens sicherhergestellt. Der besondere Vorteil dieses Verfahrens ist das vorhersagbare Echtzeitverhalten. Beim Profibus wird dies mit einem Master-Slave-Verfahren kombiniert. Die Master tauschen das Senderecht über das Token-Passing-Verfahren untereinander aus und fragen ihre Slaves individuell ab, wenn sie im Besitz des Senderechts sind.

TDMA Für die Kommunikation wird eine Periode festgelegt (TDMA-Zyklus), diese wird in eine Anzahl von Zeitschlitzen oder Zeitscheiben mit einer definierten Größe aufgeteilt. Jeder Teilnehmer erhält einen oder mehrere dieser Zeitscheiben. Innerhalb dieser Zeitscheiben besitzt der zugeordnete Teilnehmer das Senderecht.

Die zweite Gruppe der stochastischen Zugriffsverfahren zeichnen sich durch eine ereignisgesteuerte Kommunikation und ein permanentes Abhören des Kommunikationsmediums aus. Durch diesen stochastischen Zugriff sind Antwortzeiten nicht vorhersagbar. Aus diesem Grund liegt zumeist die genutzte Belastung des Kommunikationssystems auf einem niedrigen mittleren Niveau. Zu dieser Gruppe gehören CSMA-Verfahren, mit ihren Vertretern CSMA/CD (Beispiel: Ethernet) und CSMA/CA (Beispiel: CAN). Bei den CSMA-Verfahren ist jeder Teilnehmer berechtigt ohne eine explizite Zuteilung des Senderechtes auf den Bus zuzugreifen (Multiple Access). Um auf den Bus zuzugreifen, prüft der Teilnehmer zunächst ob der Bus frei ist (Carrier Sense) und unternimmt einen Senderversuch, falls der Bus von keinem anderen Teilnehmer verwendet wird. Beim Auftreten von Kollisionen unterscheiden sich die beiden Verfahren.

CSMA/CD Bei dem CSMA/CD-Verfahren werden Kollisionen erkannt (Collision Detection). Die Teilnehmer brechen ihren Sendeversuch ab und wiederholen diesen nach einer teilnehmerspezifischen Wartezeit.

CSMA/CA Bei dem CSMA/CA-Verfahren werden Kollisionen durch eine Prioritätsregel vermieden (Collision Avoidance). Vermeiden bedeutet hier, es treten Kollisionen auf, diese werden erkannt und bis auf einen Teilnehmer brechen alle anderen ihren Sendeversuch ab. Trotz mehrfachem, gleichzeitigem Zugriff auf den Kommunikationsbus bleiben die Daten eines Senders erhalten. Beim CAN-Bus wird dies durch den Identifier der Nachrichten gewährleistet.

4.1.2 Modellierung auf Ebene abstrakter Zugriffsmechanismen

Modelle auf der Ebene der abstrakten Zugriffsmechanismen eignen sich insbesondere zur Visualisierung und zur Analyse eines konkreten Buszugriffsverfahrens. Durch eine abstrakte Darstellung sind dabei die Abläufe während einer Kommunikation sehr gut zu erfassen. Somit eignen sich diese Modelle insbesondere für einen didaktischen Einsatz zur Vermittlung und Gegenüberstellung unterschiedlicher Zugriffsverfahren, wie sie in konkreten Bussystemen zum Einsatz kommen. Die Modellierung und Simulation von applikationsspezifischem Verhalten spielt an dieser Stelle eine untergeordnete Rolle. Daraus lässt sich schlussfolgern, dass sie zur Untersuchung von Gesamtsystemen aus verschiedenen Gründen nur eingeschränkt verwendbar sind. Die dargestellten Modelle lassen sich teilweise zwischen der *Level 1* und der *Level 2* einordnen, da einige protokollspezifische Funktionen berücksichtigt werden, auch wenn die Aussagekraft der Modelle über einen abstrakten Zugriffsmechanismus nicht hinausgeht.

4.1.2.1 Betrachtung von Petri-Netz-Modellen für zeitgesteuerte Protokolle: Beispiel FlexRay und TTP/C

4.1.2.1.1 TTP/C In der Diplomarbeit von Jentzsch (2003) erfolgte eine Modellierung des TTP-Protokolls mittels hierarchischer, gefärbter, zeitbewerteter Platz Transitions Netze (Petri-Netze). Da diese eine gute Strukturierung und Hierarchisierung erlauben korreliert dies mit dem Ziel, die erstellten Modelle im Lehrbetrieb einzusetzen.

In dieser Arbeit werden die Stärken und Schwächen der Modellierung mit Platz Transitions Netzen ersichtlich. Das Modell beinhaltet die grundlegende Funktionalität des TTP/C-Protokolls, und besteht aus einem TTP/C-Netzwerk mit zwei Teilnehmern sowie einem zentralen Buswächter. Die für den Cluster Startup sowie das Senden und Empfangen von Nachrichten relevanten Abläufe wurden in das Modell integriert. Durch die Verwendung von vier Hierarchieebenen ergibt sich eine übersichtliche Struktur. Aufgrund der Komplexität, der Übersichtlichkeit sowie der Simulierbarkeit gibt es innerhalb des Modells diverse Einschränkungen.

So wurde im Modell auf unabhängige Taktgeber, eine detaillierte Modellierung des Zustandes *Passive*, diverse Fehlervariablen und Informationen über die globale Zeit sowie die Paketgröße im Protokollrahmen auf Grund der Komplexität und Übersichtlichkeit verzichtet. Eine Untersuchung und Analyse insbesondere der Uhrensynchronisation und weiterer Mechanismen ist nicht möglich.

Bei der Analyse der dort entstandenen Modelle zeigt sich, dass die Größe des Modells abhängig ist von der Anzahl der Teilnehmer sowie der Anzahl der Slots zur Kommunikation. Dabei ergibt sich in dem Modell mit einem deutlich eingeschränkten

Funktionsumfang folgender Zusammenhang in Bezug auf die Anzahl von Plätzen ($N_{\text{Plätze}}$) und Transitionen ($N_{\text{Transitionen}}$):

$$N_{\text{Plätze}} = N_{\text{Teilnehmer}} \times (2 \times N_{\text{Slots}} + 39) + N_{\text{Slots}} + 8$$

$$N_{\text{Transitionen}} = N_{\text{Teilnehmer}} \times (2 \times N_{\text{Slots}} + 83) + N_{\text{Slots}} + 10$$

4.1.2.1.2 FlexRay In der betreuten Bachelorarbeit von Simon (2010) wird eine Modellierung des FlexRay Kommunikationsprotokolls unter Verwendung von einfachen hierarchischen Petri-Netzen ohne Färbung vorgestellt. Vergleicht man das entstandene Modell mit dem TTP/C-Modell von Jentzsch (2003), so ist das FlexRay-Modell auf Grund der sehr einfachen Elemente deutlich umfangreicher. Was die Vorteile der Visualisierung von höheren Netzen verdeutlicht.

Analysiert man die modellierten Protokollfunktionen, so zeigt sich, dass sich der Umfang des Modells auf die Anbindung von einfachen Anwendungsmodellen sowie auf die abstrakte Modellierung des zyklen- und segmentbasierten Buszugriffsverfahrens beschränkt. Ähnlich wie beim TTP/C-Modell wird auf die Modellierung der verteilten Zeit und der Uhrensynchronisation verzichtet und eine globale Zeit vorausgesetzt. Ein wesentlicher Unterschied zum TTP/C-Modell ist, dass die internen Protokollzustände nicht betrachtet werden.

Eine Bewertung des FlexRay-Modells hinsichtlich des Verhältnisses von Modellgröße und Detaillierungsgrad zeigt ein deutliches Defizit. Zur Modellierung des Zugriffsverfahrens auf einem sehr abstrakten Niveau wird ein recht großes Modell verwendet, was sich deutlich an der Anzahl der verwendeten Plätze ablesen lässt.

$$N_{\text{Plätze}} = 2 \times N_{\text{Teilnehmer}} + 9 \times N_{\text{StatischeSlots}} + 36 \times N_{\text{DynamischeSlots}} + 37$$

Durch Hinzunahme von zusätzlichen semantischen Elementen wie Färbung und Zeitbewertung kann die Übersichtlichkeit der Modelle zusätzlich gesteigert werden. Dieser Sachverhalt wird in der Gegenüberstellung der Petri-Netze von FlexRay und TTP/C deutlich.

Einfaches Modell des Buszugriffsverfahrens Nachfolgend wird eine alternative Modellierung des reinen Buszugriffsverfahrens präsentiert. Dieses Modell eignet sich für einen didaktischen Einsatz und bleibt bei der Modellierung auf dem abstrakten Niveau des reinen Buszugriffsverfahrens ohne Darstellung interner Zustände. Die Leistungsfähigkeit im Vergleich zu dem vorherigen Modell wird dabei beibehalten.

Darstellung des Modells In Abbildung 4.1.1 ist das resultierende Petri-Netz für das Zugriffsverfahren des FlexRay dargestellt. Dabei werden in diesem Modell lediglich die einfachen zeitlichen Abläufe berücksichtigt. Die einzelnen Netzwerkknoten und die konkrete Verwaltung des Senderechts auf Basis der Zeiteigenschaften werden zunächst nicht betrachtet. Ein einfaches Modell zur Anbindung eines Teilnehmers zeigt Abbildung 4.1.2.

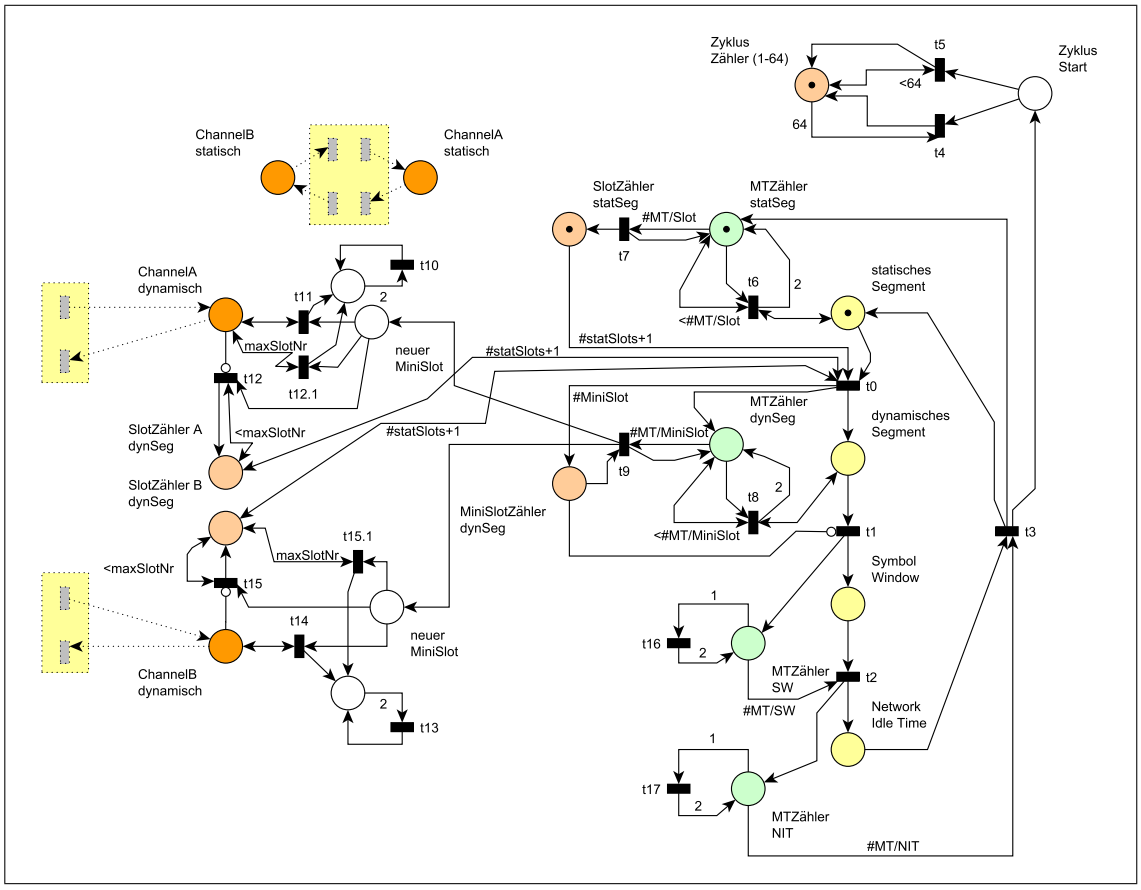


Abbildung 4.1.1: Einfaches Modell der Zyklen- und Zugriffsverwaltung des FlexRay

Zur Modellierung werden im Petri-Netz als Sonderkanten Testkanten und Inhibitorkanten verwendet, wie sie beispielsweise in (Wimmel, 2008) dargestellt werden. Die Inhibitorkante wird als Kante mit einem Kreis am Ende dargestellt. Die Testkante hat an beiden Enden einen Pfeil. Für das präsentierte Modell gilt, dass es für ein konkretes Kommunikationsszenario konfiguriert werden muss, d.h. die variablen Kantengewichte müssen abhängig von der Konfiguration des FlexRay gesetzt werden. Zu diesen Variablen zählen: die maximale Anzahl der Kommunikationsslots (*maxSlotNr*), die Anzahl der Macroticks pro Slot im Statischen Segment (*#MT/Slot*), die Anzahl der Slots im Statischen Segment (*#statSlots*), die Anzahl der Macroticks pro Minislot im Dynamischen Segment (*#MT/Slot*), die Anzahl der Macroticks im Symbol Window (*#MT/SW*) und die Anzahl der Macroticks in der Network Idle Time (*#MT/NIT*).

Das Modell kann in unterschiedliche Bereiche bzw. fünf Gruppen aufgeteilt werden. Zunächst finden sich vier Plätze (*Statisches Segment*, *Dynamisches Segment*, *Symbol Window*, *Network Idle Time*), welche die einzelnen Segmente des FlexRay-Zyklus repräsentieren. Dazu gehören jeweils vier Plätze die zur Bestimmung der jeweiligen Macrotick innerhalb der Segmente dienen (*MTZähler statSeg*, *MTZähler dynSeg*, *MTZähler SW*, *MTZähler NIT*). Aufbauend auf diesen Zählplätzen erfolgt die Festlegung der zeitlichen Eigenschaften des FlexRay-Modells, dabei wird explizit zwischen dem Statischen und dem Dynamischen Segment unterschieden, obwohl dies nicht zwingend erforderlich wäre. Die fünf Plätze *Zyklus Zähler*, *SlotZähler statSeg* , *SlotZählerA dyn-*

Seg, *SlotZählerB dynSeg* und *MiniSlotZähler dynSeg* charakterisieren den aktuellen Sende-Zeitpunkt im FlexRay. Für jeden Kanal existiert ein eigener Platz, welcher für jedes Segment jeweils den Zustand *frei* oder *belegt* repräsentiert. Diese beiden Plätze für das Statische Segment sind aus Gründen der Vollständigkeit vorhanden und können auf Anwendungsebene zur Anbindung verwendet werden. Die weiteren Plätze dienen im Allgemeinen ausschließlich zur Verwaltung. Die Modellierung der Zeit und einzelner Zeitdauer erfolgt auf Basis von Zählplätzen. Die kleinste Zeiteinheit bildet dabei die Zeiteinheit Macroticks, welche die globale Zeit in einem FlexRay-Cluster repräsentieren.

In dieser kurzen Darstellung wird ersichtlich, dass ein einzelner Knoten auf Basis der zur Verfügung gestellten Informationen nun über das eigene Senderecht entscheiden kann. Die Abbildung 4.1.2 verdeutlicht am Beispiel eines Knotens, welcher in einem Slot des Dynamischen Segment auf Kanal B etwas senden möchte, wie die Verknüpfung von Knoten und „Busmodell“ erfolgen kann. Die Bestimmung der Sendedauer bei der Übertragung wird zur Vereinfachung an dieser Stelle nicht weiter betrachtet und lediglich durch die Transition *tSendedauer_dSB* repräsentiert. Die Plätze *SW_dSB*, für einen Sendewunsch, und *Senden_dSB*, für das Andauern des Sendens eines Datenpaketes, bilden den Übergang zu dem Applikations- oder Knotenmodell, welches durch eine abstrakte Box symbolisiert wird.

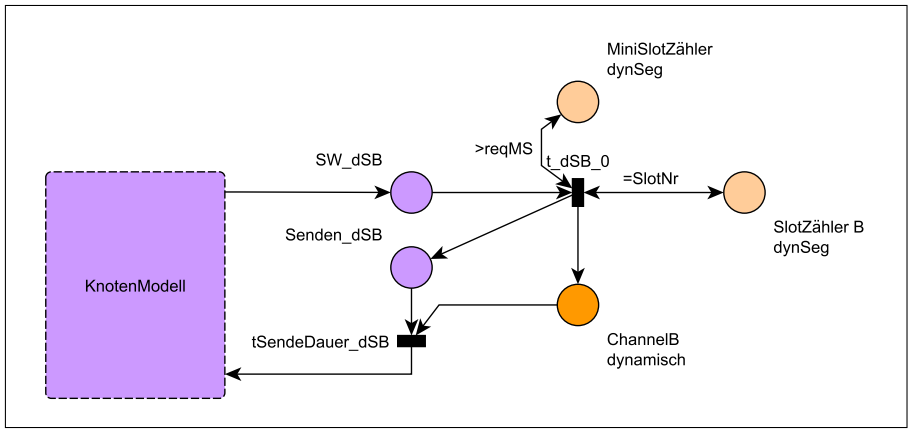


Abbildung 4.1.2: Einfaches Modell der Anbindung eines Knotenmodells zur Verwaltung des Senderechtes am Beispiel eines Slots im Dynamischen Segment

Eigenschaften und Komplexität des Modells Die Modellierung der Zeit erfolgt ebenso wie in den vorherigen Modellen auf Basis von Macroticks, welche den globalen Zustand im FlexRay darstellen. Mit der gewählten Zeitbasis wird von den internen Protokollmechanismen insbesondere der verteilten Erzeugung der gemeinsamen Zeitbasis und der Uhren-Synchronisation unter den Teilnehmern abstrahiert.

Im Gegensatz zu der Spezifikation wird durch die getrennten Slotzähler und Kanäle explizit zwischen dem Statischen und dem Dynamischen Segment unterschieden. Dies begründet sich durch didaktische Anforderungen an das Modell, welches den visuel-

len Vergleich und die Gegenüberstellung von Statischem und Dynamischem Segment vereinfacht.

Betrachtet man die Größe des Modells, so kann man diese in zwei Stufen betrachten. Die Modellierung des FlexRay-Kommunikationszyklus, welcher die Basis für die Zuteilung des Senderechtes bildet, kann mit 22 Plätzen und 20 Transitionen erfolgen:

$$N_{\text{Plätze\_nurZugriff}} = 22$$

.

In dem zweiten Schritt kann die Verteilung des Senderechtes an die Kommunikationsteilnehmer mit in die Betrachtung einbezogen werden. Neben den bisher benötigten Plätzen werden für jeden zugeteilten Sendeslot für einen Kommunikationsknoten jeweils ein Platz zur Darstellung eines Sendewunsches und des Sendevorganges benötigt. Unter Umständen können diese noch um einen Zählplatz, der die Sendedauer repräsentiert, ergänzt werden. Somit ergänzt sich das FlexRay-Modell um zwei Plätze je Sendeslot und Kanal unabhängig von den einzelnen Applikations- oder Knotenmodellen:

$$N_{\text{Plätze\_mitSendeRecht}} = 4 \times N_{\text{Slots}} + 22$$

.

Gegenüberstellung der beiden Modelle Betrachtet man nun die beiden Petri-Netz-Modelle für FlexRay zeigen sich deutliche Unterschiede in der Komplexität der Modelle. Während man in dem dargestellten einfachen Modell lediglich eine statische Anzahl von 22 Plätzen verwendet, ist die Anzahl der Plätze in dem deutlich komplexeren Modell abhängig von der Anzahl der Slots (die Abhängigkeit von Knoten ergibt sich durch die Berücksichtigung des *Symbol Window* und wird aus der Betrachtung herausgestrichen) und ist mindestens zwei mal so hoch. Die Anzahl der verwendeten Plätze für die reine Modellierung des Kommunikationszyklus bestimmt sich folgendermaßen:

$$N_{\text{Plätze}} = 3 \times N_{\text{StatischeSlots}} + 4 \times N_{\text{DynamischeSlots}} + 37$$

.

Bezieht man die Verwaltung des Senderechtes und die Zuteilung desselben zu den Teilnehmern in die Betrachtung mit ein, so bleibt diese Diskrepanz bestehen. Abhängig von der konkreten Konfiguration der Kommunikation und der Anzahl Statischer und Dynamischer Slots kann der Unterschied zwischen den Modellen deutlich größer sein (Beispiel: 4 Statische und 10 Dynamische Slots - einfaches Platz Transitions Modell etwa um den Faktor 5 geringer ca. 430 zu 78 verwendete Plätze). Zudem leidet durch die hierarchische Gliederung die Übersichtlichkeit des komplexen Modells sehr.

Abgesehen von dem Sachverhalt, dass in dem einfachen Modell das Senden innerhalb des Segmentes *Symbol Window* nicht berücksichtigt wird, gibt es im Rahmen des funktionalen Umfanges und der Abbildung der Eigenschaften des FlexRay-System der beiden Modelle keine wesentlichen Unterschiede.

4.1.2.1.3 Zusammenfassung - Modellierung mit Petri Netzen Petri-Netze eignen sich insbesondere für die Verifikation und Validierung von internen Protokollmechanismen. Die Modellierung beschränkt sich dabei auf einzelne Aspekte der Kommunikation.

In großen Systemen ist die Übersichtlichkeit mit einfachen Modellen nicht mehr gewährleistet. In komplexen Systemen deren Modelle ebenso komplex bzw. umfangreich sind, sollten im Sinne der Übersichtlichkeit zwingend höhere Netze verwendet werden (insbesondere mit Färbung und Hierarchie).

Für Kommunikationsmodelle, welche *Level 2* und *Level 3* berücksichtigen und zur Analyse von Gesamtsystemen genutzt werden, ist die Verwendung von Petri-Netzen aufgrund ihres Umfangs nur eingeschränkt nutzbar.

Zudem zeigt der Vergleich der beiden FlexRay-Modelle, dass die Aussagekraft der Modelle und die Größe der Netze sehr stark variieren können. Die Struktur und der Umfang der Modellierung und Modelle sind sehr von der konkreten Aufgabenstellung und der Erfahrung des Modellierers abhängig

4.1.2.2 Finite State Machines und SDL

4.1.2.2.1 TTP/C Lisner und Kessler (2001) stellen in ihrer Arbeit ein, die grundlegenden Eigenschaften des TTP/C-Feldbusprotokolls beschreibendes, Modell vor. Die Modellierung mit SDL erfolgt dabei auf Basis von Prozessen. Der Schwerpunkt liegt auf grundlegenden funktionalen und leistungsrelevanten Aspekten, dazu wird lediglich ein Teil der TTP/C-Funktionalität berücksichtigt. Ein TTP/C-System besteht aus einem das physikalische Verhalten modellierenden Bus-Block und mehreren Knotenblöcken, welche die gesamte Funktionalität eines Kommunikationsknotens enthalten (Timer für die Zeitbasis, Controller, Busguardian). Die Basis bildet die Microtick-Ebene der Komponenten Controller, Busguardian, Timerprozesse, CNI und Applikation. Neben diesen grundlegenden Prozessen, welche die Zugriffsrechte und die Kommunikation regeln, existieren zusätzlich Applikationsprozesse.

4.1.2.2.2 FlexRay In der betreuten Bacherlorabeit von Simon (2010) werden neben Petri-Netzen auch Statecharts für die Modellierung des Kommunikationssystems FlexRay verwendet. Das prinzipielle Vorgehen bei der Erstellung der Modelle ist identisch, daher lassen sich die resultierenden Modelle abschließend gut miteinander vergleichen. Es werden die folgenden Modellelemente verwendet: Zustand, Zustandsübergänge, Hierarchie, Nebenläufigkeit, Synchronisation und Kommunikation.

Das FlexRay-System besteht aus mehreren nebenläufigen Zuständen, zum einen die Knoten, welche wiederum voneinander unabhängig sind, und zum anderen das FlexRay-Protokoll an sich. Berücksichtigt wird an dieser Stelle nur ein sehr einfaches Knotenmodell welches ausschließlich die Fähigkeit des Buszugriffs besitzt und somit zur Simulation des Sendens von Daten verwendet werden kann. Die gemeinsame Zeitbasis aller Knoten bildet dabei der Macrotick. Zeitgenerierung und Synchronisationsmechanismen finden demzufolge im Rahmen des Modells keine Berücksichtigung.

Der Ausgangspunkt und somit der Top-Level des Protokoll-Modells bilden die vier Segmente eines Zyklus, welche jeweils durch einen Zustand dargestellt werden. Der

konkrete Zyklus wird durch eine Variable repräsentiert. An dieser Stelle sei das statische Segment beispielhaft herausgenommen, welches sich aus drei nebenläufigen Zuständen zusammensetzt. Zwei davon berücksichtigen das zeitliche Verhalten, durch Modellierung von Macroticks und Statischen Slots. Der dritte Zustand behandelt die Verwaltung des Senderechts im Statischen Segment. Analog erfolgt die Modellierung der weiteren Segmente.

Die Statechart-Modelle zeichnen sich gegenüber den Platz-Transitionsnetzen durch eine kompaktere und übersichtlichere grafische Darstellung aus. Dies ergibt sich insbesondere durch Eigenschleifen und Zählvariablen. Hieraus folgen auch die deutlichen Unterschiede in den verwendeten Elementen. Neben den Zuständen und den Zustandsübergängen könnte man die Betrachtung der Modellgröße durch die Anzahl der Variablen erweitern.

$$N_{\text{Zustände}} = 4 \times N_{\text{Teilnehmer}} + 2 \times N_{\text{DynamischeSlots}} + 12$$

$$N_{\text{Zustandsübergänge}} = 4 \times N_{\text{Teilnehmer}} + N_{\text{StatischeSlots}} + 8 \times N_{\text{DynamischeSlots}} + 88$$

4.1.2.3 Multi-Domänen-Modelle: Beispiel MLDesigner

Als Multi-Domänen-Tool wird an dieser Stelle MLDesigner ausgewählt. Auf Grund des Leistungsvermögens von Multi-Domänen-Tools insbesondere von MLDesigner lassen sich die vorhergehenden Modelle ebenso realisieren, insbesondere die Finite-State-Machines können direkt in die FSM-Domäne abgebildet werden. Ebenso lässt sich das Verhalten der Petri-Netze sehr leicht über Zählvariablen, Zustände und Ereignisse nachbilden, um das abstrakte Zugriffsverfahren zu modellieren.

Die Modellierung der Zeit und Vergabe des Senderechtes erfolgt auf Basis von Macroticks. Das abstrakte Modell arbeitet dabei auf Basis eines globalen System-Zustandes, der durch Zyklus, Slot und Macrotick repräsentiert wird. Somit wird ebenso wie bei den vorhergehenden Modellen von den internen Mechanismen des FlexRay-Protokolls abstrahiert. Es erfolgt ausschließlich eine Modellierung auf Ebene des abstrakten Zugriffsmechanismus. Die konkrete Umsetzung eines MLDesigner-Modells kann dabei in Anlehnung an das Petri-Netz in Abschnitt 4.1.2.1.2 geschehen.

An dieser Stelle wird auf die grafische Darstellung des Modells verzichtet und lediglich eine kurze verbale Beschreibung der wesentlichen Elemente vorgenommen. Auf eine ausführliche Darstellung wird verzichtet, da die einzelnen Modellbestandteile sich in den folgenden Modellierungen des FlexRay-Protokolls mit spezifischen Protokolleigenschaften in Kapitel 4.1.3 wiederfinden.

Die globalen Zählvariablen *SlotCounter Kanal A*, *SlotCounter Kanal B*, *Zyklus-Counter* und *MinislotCounter* bestimmen dabei den globalen Zustand des FlexRay-Protokolls. Die Modellkomponente *Global Clock* bestimmt im Allgemeinen auf Basis von Zeitevents und einem Macrotick-Counter die Werte der einzelnen Zählvariablen. Die Änderungen der einzelnen SlotCounter werden als Events zu den Kommunikationsteilnehmern weitergeleitet, welche eine interne Auswertung des Senderechts vornehmen und gegebenenfalls das Senden von Daten über den Bus initiieren.

Die Änderung der SlotCounter im Dynamischen Segment ist zusätzlich abhängig vom Sendeverhalten der Knoten, daher kann die Änderung durch den Knoten mit

Senderecht erfolgen. Das so modellierte Verhalten ist vergleichbar mit dem einfachen Petri-Netz-Modell und hat eine sehr kompakte Darstellung mit dem großen Vorteil der variablen, stufenweisen Verfeinerung einzelner Modellkomponenten durch flexible Substitution einzelner Komponenten.

4.1.3 Modelle mit spezifischen Protokolleigenschaften: FlexRay

Mit der Zielstellung der Unterstützung der Analyse und Modellierung unter Berücksichtigung des Gesamtsystemverhaltens werden im Weiteren für die Modelle der *Level 2* und *Level 3* ausschließlich Multi-Domänen-Tools berücksichtigt, im speziellen wird sich auf das Modellierungswerkzeug MLDesigner beschränkt. Insbesondere die einfache Kopplung, die Berücksichtigung von unterschiedlichen Abstraktionsebenen sowie Komponenten und Hierarchie werden unterstützt. Im Bereich des Cyber-Physical-Modeling in Verbindung mit partiell sehr detailreichen Modellen haben die Multi-Domänen-Tools ihre Stärke.

4.1.3.1 Modellbibliothek für die Komponente FlexRay

Die Prinzipstruktur für Kommunikationsprotokolle und angeschlossene Systeme folgt dem in Abschnitt 3.2.1 vorgestellten Basiskonzept und umfasst die Komponenten: Host (Controller des eingebetteten Systems inklusive Anwendung), Communication Controller (CC) des verwendeten Bussystems sowie dem verwendeten Bus/Channel als Übertragungsmedium selbst. Die Modelle wurden bereits auszugsweise in (Klöckner et al., 2008) veröffentlicht. Die einzelnen Komponenten lassen sich in der Abbildung 4.1.3 erkennen.

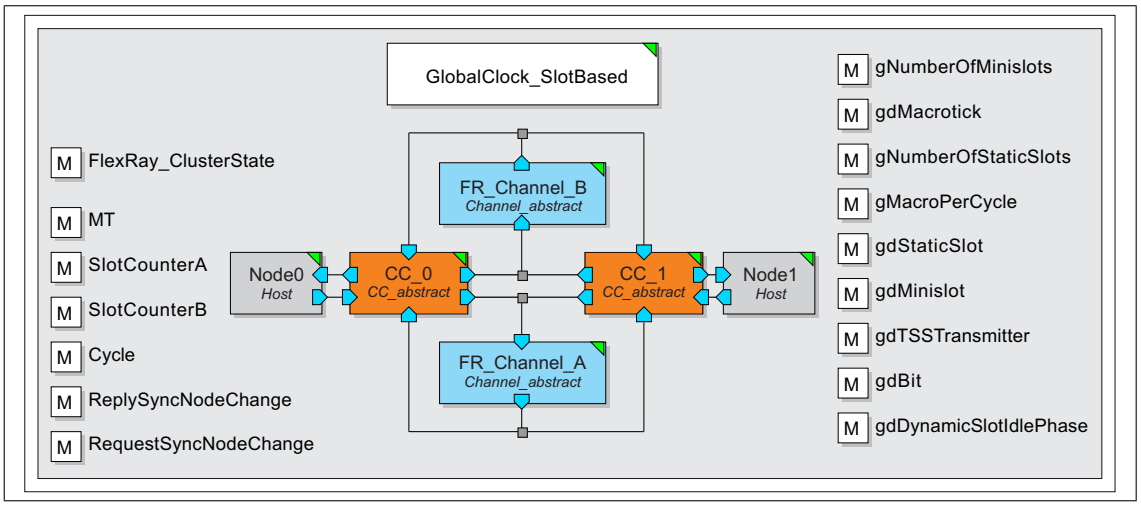


Abbildung 4.1.3: Beispiel für ein abstraktes Modell eines einfachen FlexRay-Clusters mit zwei Knoten

4.1.3.1.1 Modelleigenschaften und Charakterisierung Im nächsten Schritt werden die einzelnen Modell-Elemente definiert. Die Basis für die Modellierung bilden

die unterschiedlichen Abstraktionsniveaus. Hier werden zwei Modelltypen (abstraktes und detailliertes FlexRay-Modell) mit insgesamt vier unterschiedlichen Kategorien für die konkrete Modellierung festgelegt. Das detaillierte Modell arbeitet auf Basis eines bitbasierten Datenaustausches, die grundlegende Zeiteinheit auf Protokollebene ist somit Sample Clock bzw. Microtick. Dies sind innerhalb der FlexRay-Spezifikation die kleinsten definierten Zeiteinheiten. Auf dieser Ebene des detaillierten Modells können daher die Einflüsse von Systemparametern exakt untersucht werden.

Im Gegensatz dazu wurden für das abstrakte Modell drei unterschiedliche Abstraktionsgrade bzw. Verfeinerungen festgelegt, um eine Spezialisierung auf verschiedene Problemstellungen zu ermöglichen. Die Unterschiede der einzelnen Modelle liegen im Bereich der Zeitrepräsentation:

Zeitberechnung auf Basis von Macroticks mit einem globalen Zustand Die Macroticks repräsentieren die globale gültige Zeit innerhalb eines FlexRay-Clusters. In der ersten Variante erfolgt die Berechnung der Zeit auf Basis der Macroticks. Unter dem globalen Zustand wird verstanden, dass jeder Knoten im System über den aktuellen Zustand der Zeit, also den so genannten Mactrotick Counter, informiert ist.

Zeitberechnung auf Basis von Macroticks mit einem lokalen Zustand Der aktuelle Zustand des Macrotick Counter ist bei dieser Variante nur innerhalb der Global Clock sichtbar. Nur ausgewählte Änderungen werden an alle Knoten weitergegeben, dies ist zum Beispiel der Fall wenn ein neuer Slot startet.

Slotbasierte Zeitbasis Bei dieser Variante werden die Macroticks nicht beachtet. Eine Berechnung der Zeit erfolgt nur auf Basis des aktuellen Slots.

Eine zentrale Rolle bilden die Schnittstellen zwischen den einzelnen Elementen. Auf der obersten Ebene ist hier die Schnittstelle zwischen Host und CC, deren Realisierung ist Service-orientiert. Dabei stellt die eigentliche Schnittstelle das Modul CHI (Controller-Host-Interface) dar, welches als Teil des Communication Controller Modells in Abbildung 4.1.4 zu erkennen ist. Dieses realisiert die angebotenen Services und ermöglicht eine einfache Portierung zwischen einzelnen FlexRay-Modellen. Auch bei der zweiten Schnittstelle zwischen CC und Channel/Bus ist eine Austauschbarkeit gewährleistet.

Um die Austauschbarkeit noch weiter zu erhöhen, erfolgt die Konfiguration der Kommunikationsmodule über die angebotenen Services. Damit ist diese Aufgabe der Anwendung (Host) zuzuordnen. Protokollelemente unterschiedlicher Abstraktionsgrade können ausgetauscht werden, ohne dass Änderungen im Bereich der Konfiguration oder der Anwendung notwendig sind.

4.1.3.1.2 Modell auf Applikationsebene (Host-Modell) Die Elemente stellen Basismodule zur Verfügung, die zur Initialisierung, zum Senden und Empfangen genutzt werden können. Prototypisch wird eine einfache Anwendung umgesetzt, mit der ein Zugriff auf den CC demonstriert wird. Die flexible Nutzung von Anwendungsmodellen ermöglicht eine Vielzahl verschiedener Abstraktionsstufen und Analysemöglichkeiten, dies wird auch durch die einheitliche Schnittstelle zu den unterliegenden Modellen unterstützt.

4.1.3.1.3 Abstraktes Modell Die Basis für das abstrakte Modell bildet die Annahme, dass die Kommunikation framebasiert erfolgt. Zwischen den Teilnehmern werden nicht einzelne Bits sondern komplette Datenframes/Nachrichten ausgetauscht. Dadurch ist es möglich, die Zeitbasis zu variieren und eine Beschleunigung in der Simulationszeit zu erreichen. Die Erzeugung einer globalen Zeit wird durch das Modul Global Clock (GC) realisiert. Dieses wird über Variablen mit anderen Modulen gekoppelt. Jeder CC wird über einen Zustandswechsel benachrichtigt. Abhängig vom Abstraktionsgrad sind diese Zustandswechsel unterschiedlich definiert. Die Benachrichtigung erfolgt zum Beispiel auf Basis von Macroticks oder Slots. Für die Realisierung unterschiedlicher Zeitbasen ist ausschließlich das Modul GC verantwortlich. Der abstrakte CC, vereinfacht in Abbildung 4.1.4 dargestellt, besteht aus Modulen zum Senden (SendtoChannel) und Empfangen von Daten (ProcessRecData), einem Modul zum Abgleich des internen Zustandes mit dem globalen Zeitgeber (UpdateStatus) und dem Schnittstellenmodul CHI (Controller Host Interface).

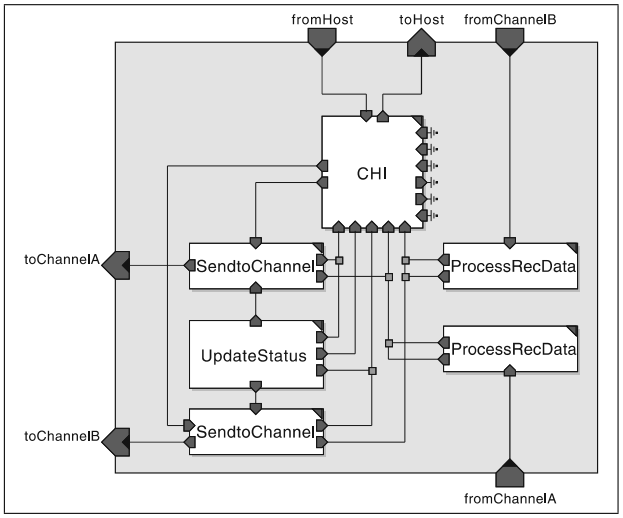


Abbildung 4.1.4: Interne Struktur des abstrakten FlexRay CC-Modells

Das Modul CHI realisiert die angebotenen Services und steuert den Zugriff auf das Protokoll von Seiten der Anwendung. Die Umsetzung der Services orientiert sich an der Protokoll-Spezifikation. Einige wenige Funktionalitäten wurden mit gewissen Einschränkungen umgesetzt. Dies betrifft die Funktionen Timer und Managementvector, diese sind für allgemeine Untersuchungen vernachlässigbar und können bei Bedarf integriert werden. Eine Hauptaufgabe des CHI ist die Verwaltung der Konfiguration sowie der Datenspeicher für das Senden und Empfangen.

Das Modul für den Channel hat die Aufgaben der Verzögerung von Nachrichten und die Verwaltung des Buszustandes, dieser ist in Zusammenarbeit mit der GC zur Berechnung der einzelnen Slots im Hinblick auf das Dynamische Segment wichtig. Innerhalb des Moduls wird ein einfaches Fehlermodell realisiert. Diese prototypische Umsetzung markiert auf Basis einer Wahrscheinlichkeitsverteilung fehlerhafte Nachrichten.

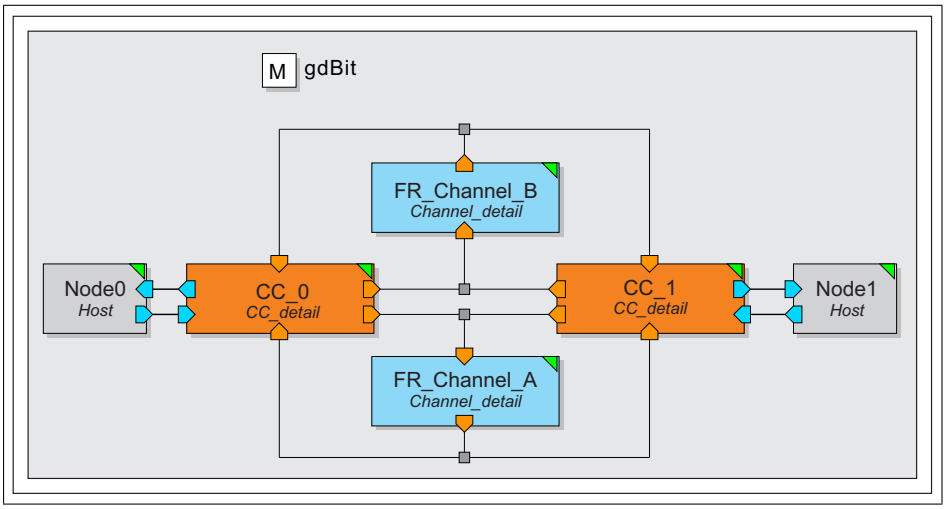


Abbildung 4.1.5: Beispiel für ein detailliertes Modell eines einfachen FlexRay-Clusters mit zwei Knoten

4.1.3.1.4 Detailliertes Modell Die Verarbeitung innerhalb des detaillierten Modells erfolgt auf Bitebene. Die einzelnen Protokollmechanismen werden mit einem hohen Detaillierungsgrad (ohne Abstraktion der Spezifikation) abgebildet. Aufgrund dieses hohen Detaillierungsgrades werden in das Kommunikationsmodul in Abbildung 4.1.8 zwei zusätzliche Elemente eingefügt, dies sind die Module BusDriver (BD), zur Realisierung des bitbasierten Datenzugriffs, sowie PowerSupply (PS). Somit wird die Struktur des FlexRay direkt im Modell abgebildet. Die Modelle des detaillierten FlexRay-Modells sind in enger Kooperation mit dem Projektpartner (Mission Level Design GmbH) im Verbundprojekt („Modellbibliothek für das Kommunikationssystem FlexRay“) entstanden, dies betrifft insbesondere die Realisierung einzelner interner Protokoll-Prozesse.

Innerhalb des Communication Controllers sind die einzelnen Protokollfunktionen entsprechend der FlexRay-Spezifikation abgebildet. In Abbildung 4.1.6 ist die interne Struktur dargestellt, hier finden sich die in der FlexRay-Spezifikation mittels SDL definierten Prozesse (POC, MTG, CSP, CSS, MAX, CODEC und FSP) sowie das CHI, welches das Interface für den Host bildet. Die einzelnen Prozesse werden mittels FSMs umgesetzt.

Das Channel-Modell verwaltet den aktuellen Buszustand (Bitwert) und sendet den aktuellen Zustand an die BusDriver.

Das detaillierte Modell visualisiert alle internen Zustände und Signale des Protokolls und erlaubt somit Untersuchungen welche diese internen Mechanismen berücksichtigen. Durch die große Anzahl von Signalen bzw. Events ist die Simulation entsprechend aufwändig. Neben diesen umfangreichen Analysen sind durch die ausführliche Modellierung direkte Codegenerierung denkbar, wie diese in (Baumann, T. and Hauguth, M. and Salzwedel, H., 2007) vergleichbar ausgeführt werden.

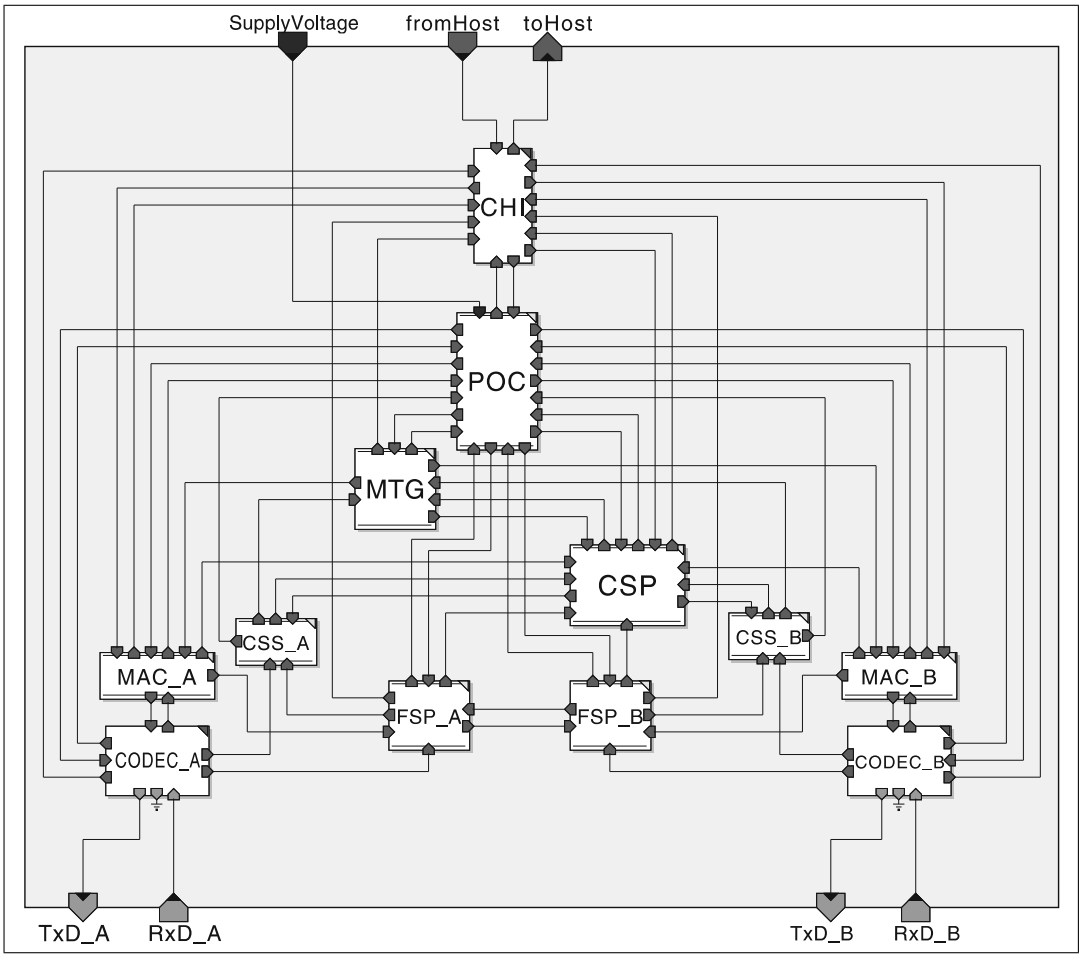


Abbildung 4.1.6: Interne Struktur des detaillierten FlexRay CC-Modells

4.1.3.1.5 Eigenschaften der FlexRay-Modelle Eine wichtige Eigenschaft der Modelle im Hinblick auf die Untersuchung von Systemen ist die Simulationszeit. Die Tabelle 4.1 zeigt die Laufzeiten der Simulation eines Systems unter Verwendung unterschiedlicher Kommunikationsmodelle, die sich durch ihren Abstraktionsgrad unterscheiden. Das simulierte System hat im Falle der abstrakten Modelle eine Laufzeit von zwei Sekunden und im Falle der detaillierten Modelle eine Laufzeit von einer Sekunde.

Tabelle 4.1: Gegenüberstellung der FlexRay-Modelle bzgl. ihrer Simulationszeit

Modell-Typ	**Realzeit**	**Laufzeit Simulation**	**Verhältnis**
Abstrakt MT global	2 Sekunden	400 Sekunden	1:200
Abstrakt MT lokal	2 Sekunden	240 Sekunden	1:120
Abstrakt Slot	2 Sekunden	60 Sekunden	1:30
Abstrakt experimentell	2 Sekunden	20 Sekunden	1:10
Detail	1 Sekunde	4960 Sekunden	1:4960
Detail experimentell	1 Sekunde	2410 Sekunden	1:2410

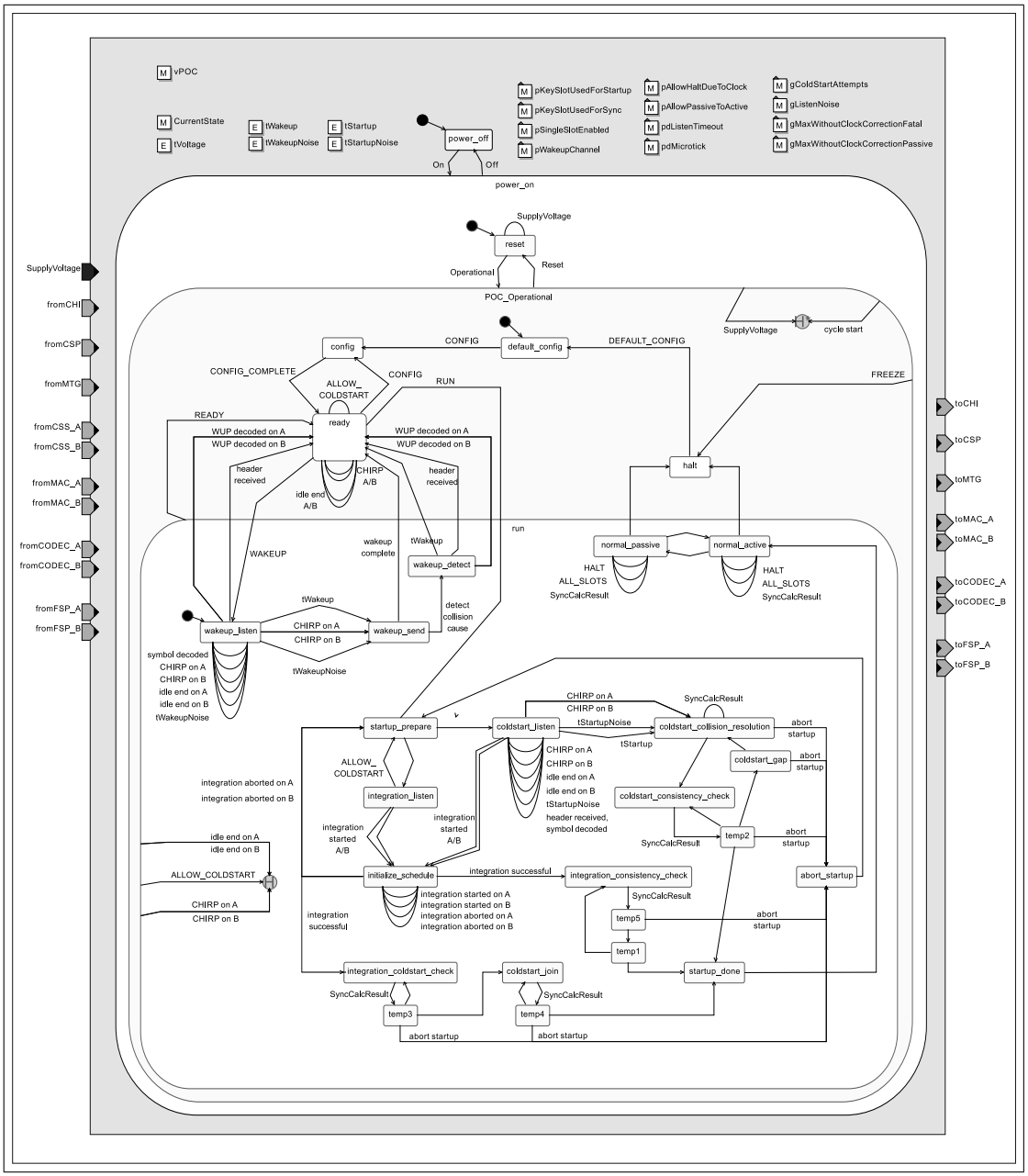


Abbildung 4.1.7: FSM-Modell des internen Protokollprozess POC

Neben den vier definierten Modellvarianten, werden in den Vergleich der Laufzeiten zwei experimentelle CC-Modelle einbezogen. Bei diesen wurden die vorgefertigten Module teilweise durch spezielle Primitive ersetzt. Die Ergebnisse zeigen, dass sich eine deutliche Verbesserung der Laufzeit durch die Verwendung von spezifischen Primitiven ergibt. Somit können zur Modellierung kompaktere Modellkomponenten zur Verfügung gestellt werden. Die so erstellten Modelle besitzen allerdings Einschränkungen hinsichtlich der beobachtbaren Ereignisse und somit auch der beobachtbaren System-Eigenschaften.

Die Ergebnisse der Untersuchungen können nur als grobe Charakterisierung ange-

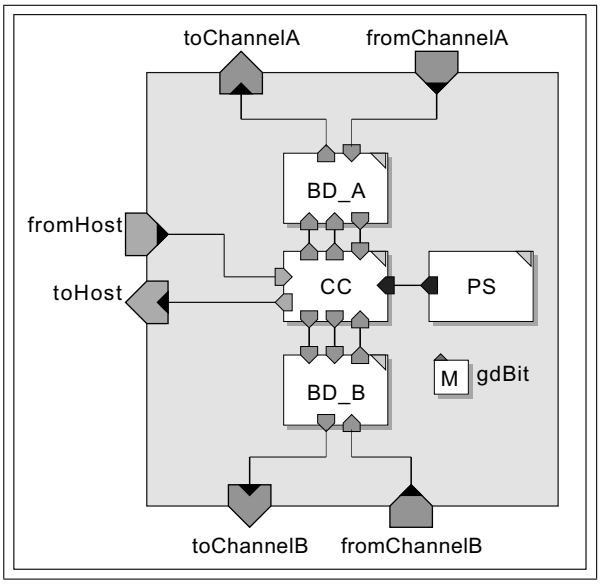


Abbildung 4.1.8: Interne Struktur des detaillierten Kommunikationsmoduls

sehen werden, da diese an konkreten Szenarien ermittelt wurden. Die Laufzeit wird nicht ausschließlich von der Abstraktion beeinflusst, sondern ist auch von weiteren Faktoren wie beispielsweise Anzahl der Knoten, Anzahl von Kommunikationsslots und Dauer des Kommunikationszyklus abhängig. Diese wurden bei den Testszenarien nicht variiert.

Für konkrete Analysen können abhängig von der konkreten Fragestellung und den damit verbundenen Anforderungen spezifische Adaptionen der Basismodule vorgenommen werden zur Optimierung oder Verbesserung der Simulationseigenschaften. Dies kann auch in einer Steigerung des Abstraktionsgrades resultieren. Durch die Modellstruktur ist eine intuitive Erweiterung und Adaption einzelner Komponenten sehr einfach möglich.

4.1.3.1.6 Modelle im Vergleich mit einem realen Aufbau Zur Prüfung der Funktionalität des Modells wird eine Simulation eines kleinen Systems durchgeführt und die beobachtbaren Eigenschaften mit einem realen Systemaufbau verglichen, dieser Vergleich erfolgt anhand der simulierten Daten und der echten Daten des FlexRay-Cluster.

Der FlexRay-Cluster wird mit Hilfe eines „FlexEntry“-Starterkits (TZ Mikroelektronik, 2006), bestehend aus zwei FlexRay-Knoten, sowie einem FlexRay Interface VN3600 (Vector Informatik, 2013), welches mit Hilfe des Tools CANoe die Analyse des Busverkehrs auf einem PC ermöglicht. Zur Konfiguration des Hardware-Aufbaus wird die Software FlexConfig (Mikroelektronik, 2006) eingesetzt. Das Simulationsmodell (abstraktes CC Modell) wird analog zum Hardware-Aufbau erstellt und konfiguriert. Die Ergebnisse der Untersuchungen zeigten eine erfolgreiche Abbildung des realen Verhaltens im Modell. In der mitbetreuten Arbeit von Hohmann (2007) werden vergleichbare Untersuchungen dargestellt.

4.1.3.2 Meta-Modell der Modellbibliothek für FlexRay

Nach der Darstellung der einzelnen entwickelten Modellkomponenten für das Kommunikationsprotokoll FlexRay werden diese abschließend in das Meta-Modell eingeordnet. In Abbildung 4.1.9 ist eine Übersicht dieser Einordnung dargestellt, welche auf dem SCS-Meta-Modell basiert. Die Basiselemente des Meta-Modells sind in der Grafik eingefärbt.

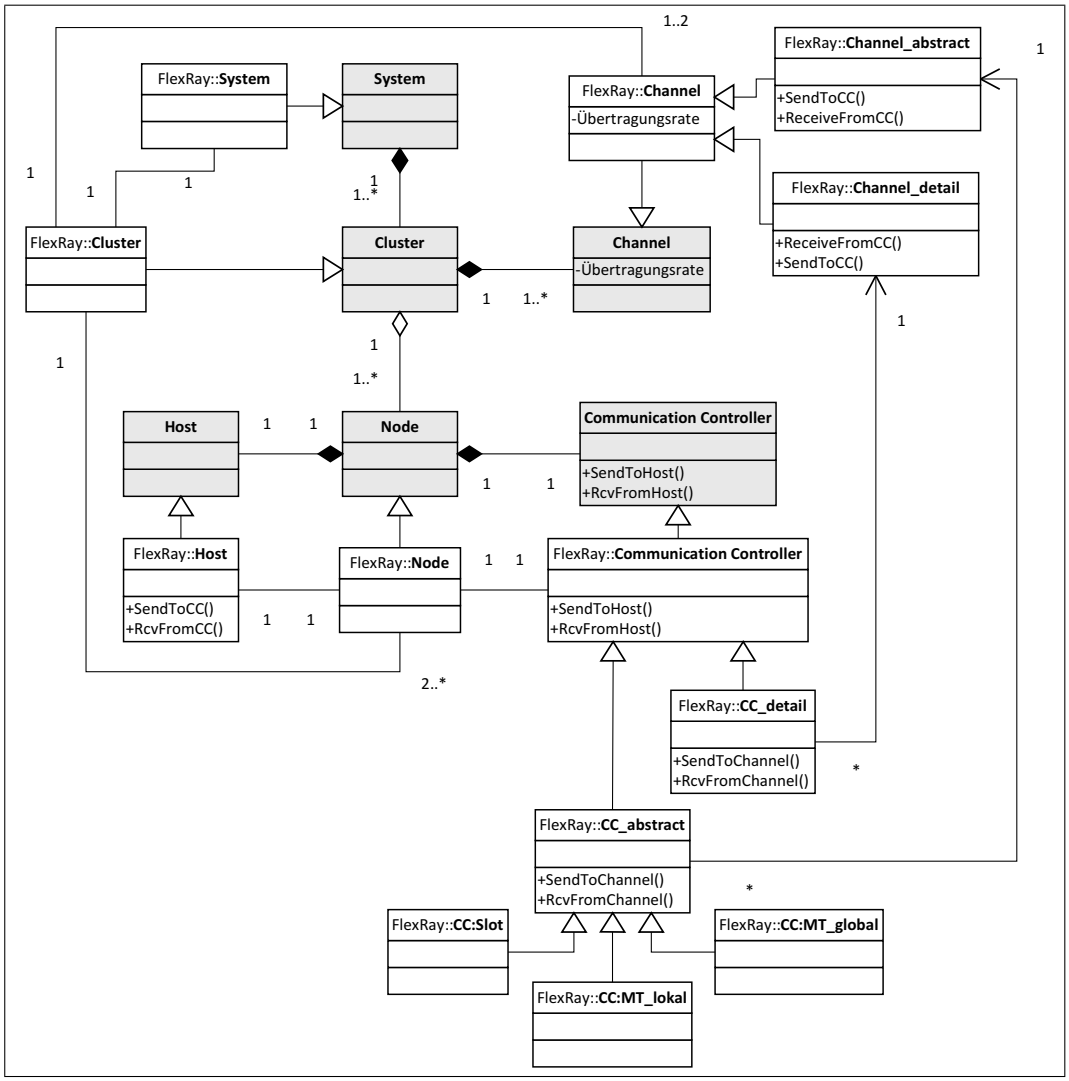


Abbildung 4.1.9: Meta-Modell der Modellbibliothek für FlexRay

Für die einzelnen Basiselemente existiert jeweils ein Repräsentant für den FlexRay. Der *FlexRay::Cluster* (*Cluster*) bildet eine logische Struktur und gruppiert die einzelnen Knoten zu einem Kommunikationsverbund. Für den *FlexRay::Channel*, welcher einer abstrakten Klasse entspricht existiert keine entsprechende Modellkomponente, sondern es gibt zwei Repräsentationen für die beiden Abstraktionstypen (abstrakt und detailliert). Der *FlexRay::Channel_abstract* arbeitet auf der Ebene von Frames und der *FlexRay::Channel_detail* arbeitet auf der Ebene von Bits. Diesen Kompo-

nenten für das Element Channel sind jeweils Communication Controller zugeordnet, welche den gleichen Abstraktionstyp unterstützen: dies sind *FlexRay::CC_detail* und *FlexRay::CC_abstract*. Für den Letzteren (*FlexRay::CC_abstract*), welcher analog zum *FlexRay:Channel* einer abstrakten Klasse entspricht, existieren jeweils drei Realisierungen. Diese Modellrealisierungen entsprechen den drei unterschiedlichen Abstraktionsgraden für die abstraktere Modellierung des FlexRay-Protokolls.

Der *FlexRay::Node* beinhaltet einen *FlexRay::Communication Controller* sowie einen *FlexRay::Host* welcher grundlegende Funktionen zum Senden und Empfangen von Daten über den *FlexRay::Communication Controller* besitzt. Über das Meta-Modell kann eine Strukturierung und Charakterisierung der einzelnen Modellkomponenten vorgenommen werden, zudem beschreibt es die Kopplung der einzelnen Komponenten zur Erstellung eines Systems zur Simulation.

4.1.4 Modelle mit spezifischen Protokolleigenschaften: CAN

4.1.4.1 Modellbibliothek für die Komponente CAN

Zur Modellierung von automobilen Netzwerken ist neben dem FlexRay auch der CAN-Bus von erhöhter Relevanz. Zur Modellierung auf Cluster-Ebene, welche insbesondere die Interaktion zwischen CAN und FlexRay sowie den Technologie-Wechsel von CAN nach FlexRay adressiert, wird zur Analyse der Verhaltenseigenschaften ein abstraktes, nachrichten-basiertes Modell vorgestellt. Zusätzlich werden Möglichkeiten zur Modellierung von Fehlern und die Einstreuung von Fehlern (Fault Injection) innerhalb von abstrakten Modellen demonstriert.

Die entwickelten CAN-Modelle verwenden die MLDesigner-Domänen DE und FSM, die sich besonders für vernetzte eingebettete Systeme eignen. Für detaillierte Untersuchungen, welche physikalische Eigenschaften berücksichtigen, können die Modelle im Sinne des Multi-Domänen-Ansatzes um entsprechende Funktionen ergänzt werden. Die entwickelten Modelle wurden bereits auszugsweise in (Müller, Klöckner & Fengler, 2011) veröffentlicht.

4.1.4.1.1 Nachrichten-basiertes CAN-Modell Innerhalb der Modellierung wird von der Bitübertragungsschicht abstrahiert und das CAN-Telegramm (CAN-MAC-Frame) als elementare Einheit bei der Informationsübertragung eingesetzt. Innerhalb der Discrete Event Domäne wird die Übertragung von Nachrichten durch Generierung und Verbrauch von Events modelliert, das Frame und dessen Inhalt werden als komplexe Datenstruktur einem solchen Event angefügt.

Die Übertragung einer Nachricht erfolgt mit einem Ereignis und damit auch ohne Zeitverzögerung. Die Zeitverzögerung der bitweisen Übertragung muss explizit mit in das Modell einfließen. Zusätzlich müssen die bitbasierten Mechanismen und Effekte des CAN berücksichtigt werden, dies betrifft insbesondere das Buszugriffsverfahren (Bus Arbitration) aber auch die Fehlerbehandlung sowie Mechanismen zur Einstreuung von Fehlern. Im Rahmen der nachrichten-basierten Modelle müssen bit-basierte Effekte berücksichtigt werden.

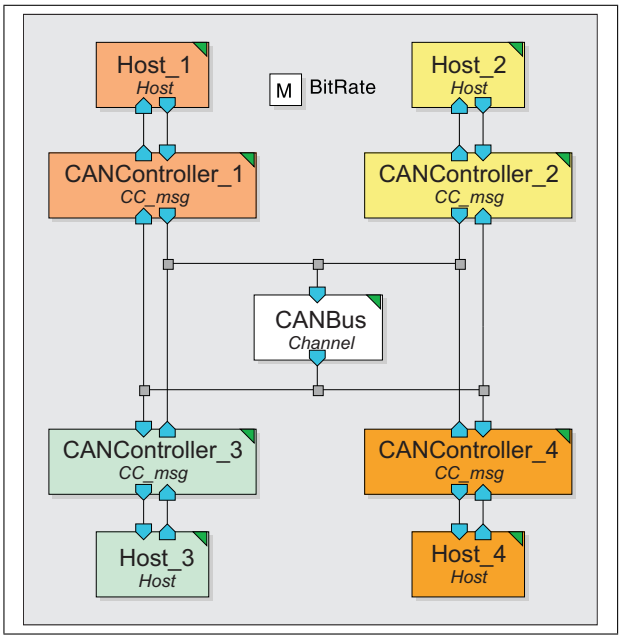


Abbildung 4.1.10: Beispiel zur Struktur des CAN-Modells

Die Struktur des Modells folgt konsequent dem Modellkonzept, dies zeigt das in Abbildung 4.1.10 dargestellte Beispiel bestehend aus vier CAN-Knoten die über den Bus verbunden sind. Jeder Knoten (Node) besteht aus einem Host-Element und einem Communication Controller. Letzterer beinhaltet die CAN-Protokoll Funktionalitäten der LLC-Ebene (Logical Link Control) und MAC-Ebene (Media Access Control). Anders als bei den FlexRay Beispielen sind an dieser Stelle die einzelnen Knoten eingefärbt, die Zuordnung zu der in Kapitel 3.2.1 vorgestellten Basisarchitektur (SCS) der Modellierung ist folgendermaßen: in der Abbildung entsprechen dem SCS-Element Channel der Block CANBus, dem SCS-Element Communication Controller die Blöcke CANController_1 bis CANController_4 und dem SCS-Element Host die Blöcke Host_1 bis Host_4.

Für die framebasierte Abstraktion des Protokolls ist das Channel-Element von zentraler Bedeutung, da wesentliche Eigenschaften der Übertragung und Funktionalitäten des CAN-Protokolls auf der bitweisen Manipulation und Auswertung des Datenstroms beruhen. Der Zugriff auf das Busmedium erfolgt dabei nach dem CSMA/CA Prinzip. Das CAN-Bus-Element bringt bit-basierte Effekte in das abstrakte nachrichten-basierte Modell, insbesondere Mechanismen der Fehlererkennung, Fehlerwahrscheinlichkeit u.ä.

4.1.4.1.2 CAN Communication Controller Der Communication Controller beinhaltet im Wesentlichen die in der CAN2.0-Spezifikation (Robert Bosch GmbH, 1991) festgelegte Funktionalität der MAC-Schicht (Buszugriff per Carrier Sense, lokale Fehlerbehandlung, knoteninterne Fehlerzustände), die hauptsächlich in Form von FSMs umgesetzt wurde. Ergänzt wird der Communication Controller durch eine Implementierung der LLC-Schicht mit Basic-CAN-Eigenschaften. Somit zerfällt er, wie in Abbildung 4.1.11 zu erkennen, in zwei Teile. Zum einen das MAC-Modul, dessen interne

Modellstruktur in Abbildung 4.1.12 dargestellt ist, zur Steuerung der Kommunikation und das LLC-Modul welches die Dienste dem Host zur Verfügung stellt.

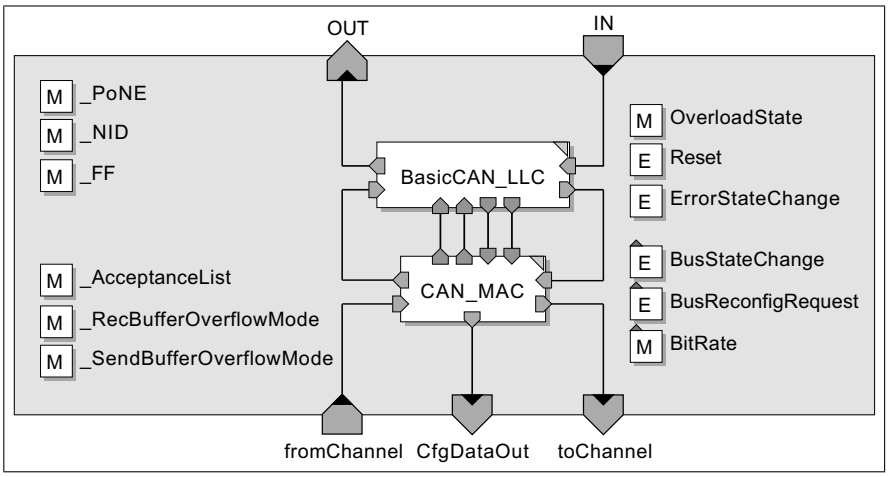


Abbildung 4.1.11: Modell des CAN Communication Controller

Auf Modellebene lassen sich Knoten- und Clusterparameter einstellen. Zu den Clusterparametern zählen das Frame Format (Standard oder Extended Frame Format) und die Bitrate. Die Knotenparameter lassen sich in zwei Gruppen aufteilen, zum einen Parameter für die Kommunikation insbesondere die Definition der Nachrichten (Message-Identifier und Datengröße) sowie Empfangsfilter und zum anderen Parameter für die Fault Injection und Fehlerbehandlung. Zur Kennzeichnung eines fehlerhaften Knotens werden zusätzlich übergeordnete Knoten-Identifier eingefügt. Zudem gibt es zwei unterschiedliche Parameter zur Fehlergenerierung und Identifikation: die Wahrscheinlichkeit eines Knotens ein fehlerhaftes Frame zu erzeugen („PoNE“ - Probability of Node Error) und die Wahrscheinlichkeit des Erzeugens einer falschen Fehlermeldung („PoCFE“ - Probability of Creating a Fake Error).

Tabelle 4.2: MAC2ChanFrame

MAC2ChanFrame	
FL - FrameLength	Framegröße in Bit zur Bestimmung der Übertragungsdauer
ID - Identifier	CAN Message Identifier
FF	Frame Format (Extended oder Standard Frame)
RTR	Remote Transmission Request Bit
Data	Übertragungsdaten
PoNE	Wahrscheinlichkeit eines Knotens ein fehlerhaftes Frame zu erzeugen

Für den Datenaustausch zwischen Communication Controller (CC) und Channel werden innerhalb der abstrakten Modellierung zwei richtungsabhängige Formate definiert: MAC2ChanFrame und Chan2MACFrame. In Tabelle 4.2 ist der Aufbau des

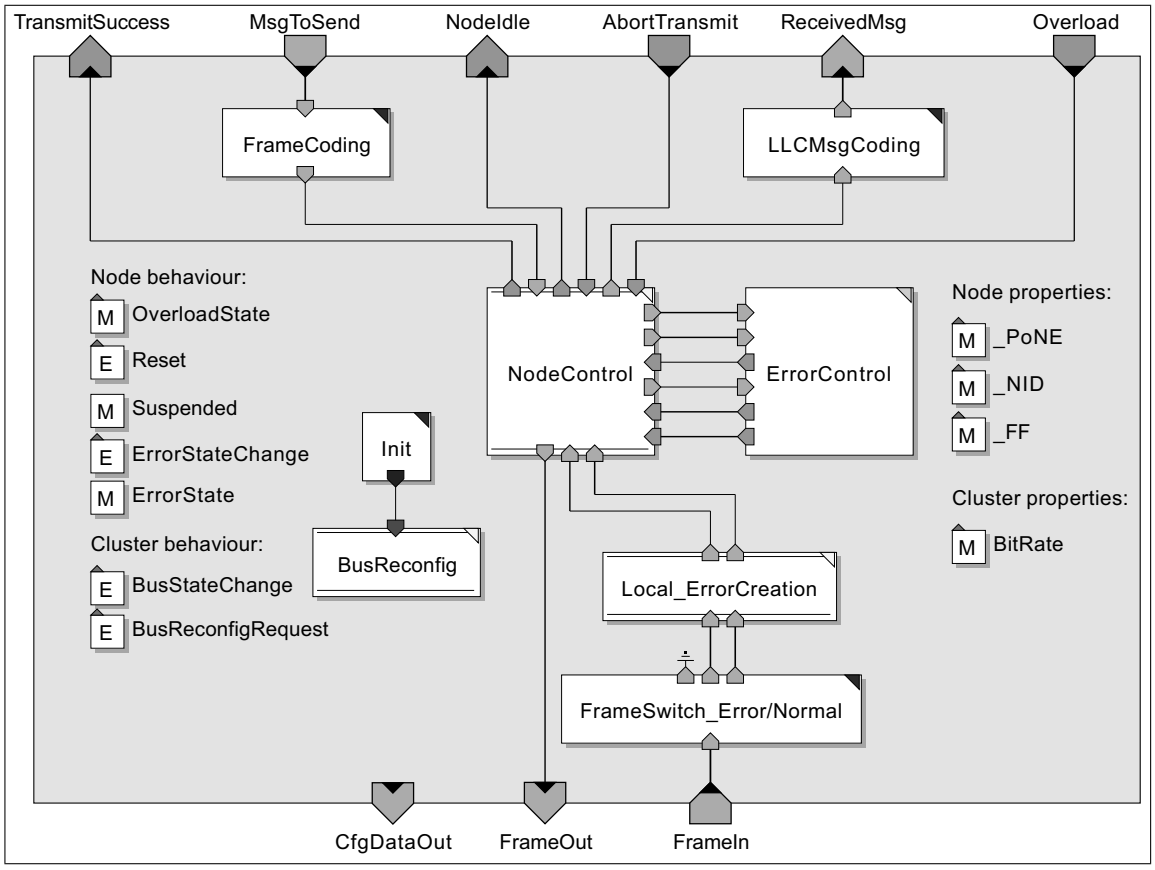


Abbildung 4.1.12: MAC-Modul des CAN Communication Controller Modells

Tabelle 4.3: Chan2MACFrame

Chan2MACFrame			
Fehlerfreies Frame		Fehlerhaftes Frame	
FL	Framegröße in Bit zur Bestimmung der Übertragungsdauer	EFL - Error Frame Length	Angepasste Frame Länge abhängig von Zeitpunkt der Fehlererkennung
ID	CAN Message Identifier		
FF	Frame Format (Extended oder Standard Frame)		
RTR	Remote Transmission Request Bit		
Data	Übertragungsdaten	FD - First Detector	Knoten der den Fehler erkannt hat

vom Knoten erzeugten Frames dargestellt. Der Channel empfängt dieses Format, wandelt es in ein korrektes oder fehlerhaftes Frame um (Tabelle 4.3) und leitet es an die Knoten weiter. Die Kommunikation zwischen CAN CC und Host erfolgt durch vom LLC realisierte Dienste: Senden und Empfangen von Nachrichten, Fehlerzustand und Reset Node.

4.1.4.1.3 CAN Channel Element Das Channel-Element simuliert die Auswirkungen der globalen bitbasierten Vorgänge auf die übertragenen Frames. Auf der einen Seite ist dies die Arbitrierung (Collision Resolution), die anhand der Eigenschaften der zu einem Simulationszeitpunkt eintreffenden Frames entschieden werden kann. Auf der anderen Seite geht es um die Kontrolle des Zeitverhaltens der Frameübertragung. Das CAN-Protokoll arbeitet ereignisgesteuert, daher ist das Zeitverhalten der Übertragung, und damit die Busauslastung, von der Länge der übertragenen Frames abhängig. Da ein Fehler die Übertragung abbricht, kommt der Modellierung der Fault Injection eine besondere Bedeutung zu, da sie einen direkten Einfluss auf die Busauslastung hat.

So ergeben sich die zentralen Aufgaben des Channel Moduls wie Frame Transformation, Darstellung eines globalen Buszustands, Steuerung des Buszugriffs (Bus Arbitration), Modellierung der Übertragungszeit (Verzögerung von Nachrichten) sowie Fehlererzeugung und Fehlerbehandlung.

Steuerung des Buszugriffs Das bitweise CSMA/CA-Verfahren wird realisiert, indem vom Channel-Element alle Frame-Partikel mit identischem Zeitstempel verglichen werden und alle mit Ausnahme des dominanten Frames verworfen werden. Neben dieser Steuerung des Buszugriffsverfahrens ist die Ermittlung der Übertragungszeit und somit auch der Busauslastung von zentraler Bedeutung, hierunter fallen wegen ihrem Einfluss auch die Mechanismen des Bitstuffing und Fehlerbehandlung. Die Bestimmung der Übertragungszeit erfolgt auf Basis der Frame-Länge (Frame Length FL oder Error Frame Length EFL), der Anzahl von Stuffbits und der Fault-Injection.

Bestimmung der Stuffbits Die Anzahl der Stuffbits hat einen direkten Einfluss auf die Größe der Nachricht und somit auf die Übertragungsdauer. Das Simulationsmodell basiert auf einer a-priori Bestimmung der Verteilung der Stuffbits. Ein konkretes Verteilungsmodell kann hierbei gewählt oder applikationsspezifisch erstellt werden. Für die Verteilung der Stuffbits gibt es unterschiedliche Ansätze. In (K. W. Tindell, Hansson & Wellings, 1994) wird eine worst-case-Annahme von $(34 + 8 * DLC)/5$ in einem Standard-CAN-Frame angegeben. In (Rauchhaupt, 1994) findet sich eine Wahrscheinlichkeitsverteilung für das Auftreten von einem bis zehn Stuffbits sowie einer durchschnittlichen Anzahl Stuffbits im Datenbereich eines Frames. In (Nolte, Hansson, Norström & Punnekkat, 2001) wird ein probabilistisches Modell für die Schätzung von Stuffbits eingeführt.

Basierend auf diesen Ideen erfolgt ein Ansatz bei dem Frames anhand von drei Attributen klassifiziert werden: die ID dessen Länge den Frame-Typ Standard- oder Extended bestimmt; das RTR-Bit, dass die Daten-und Remote Frames unterscheidet, und die Datenlänge DLC, welche die Anzahl der Datenbytes (eins bis acht) angibt. Die Kombination dieser Kriterien resultiert in sechszehn Klassen für alle Längen von Standard- und Extended Daten-Frames sowie zwei Remote Frame-Klassen.

Die a-priori-Bestimmung der Stuffbits kann rein statistisch oder bei Vorhandensein von realen Daten anwendungsabhängig durchgeführt werden. Eine resultierende Verteilung wird in das Frame-basierte CAN-Szenario-Modell eingefügt. Ankommende Frames werden so klassifiziert und auf Basis der Stuffbit-Verteilung eine realistische

Frame-Länge und Sendedauer bestimmt. Eine Variation verschiedener Methoden zur Bestimmung der Stuffbits ist durch Substitution und Parametrierung möglich.

Mechanismen zur Fehlerbehandlung und Frame-Größe Auftretende Fehler werden in zwei Hauptklassen unterteilt CRC-Fehler und Bit-Fehler. In letzterer Klasse werden Bit-Monitoring-Fehler, Bit-Stuff-Fehler und Frame-Fehler unterschieden. CRC-Fehler werden von jedem Knoten nach einer fehlerhaften CRC-Prüfung erkannt. Bei einem Bit-Fehler wird ein Unterschied zwischen erwartetem und tatsächlichem Bus-Pegel erkannt. Dies kann als Monitoring-Fehler vom Sender oder von allen Knoten als Verletzung der Bit-Stuff-Regel (Bit-Stuff-Fehler) oder des Frame-Formats (Frame-Fehler) erkannt werden.

Gleichzeitig wird ein Frame in vier Segmente (G1-G4) eingeteilt, in diesen sind globale Fehler erkennbar, welche durch Transceiver Fehler oder Kanalstörungen hervorgerufen werden. Den Segmenten werden die Fehlertypen zugeordnet. Zusätzlich müssen Fehler betrachtet werden, welche durch einen lokalen Empfänger-Fehler hervorgerufen werden. Für diese lokalen Fehler erfolgt eine Aufteilung in drei Segmente (L1-L3). Weiterführende Informationen zur Modellierung der Fehler und der Fehlerklassifikation finden sich in (Müller, Klöckner & Fengler, 2011). Die Fehler-Klassifikation dient zur Darstellung der CAN Fehlerbehandlung auf Nachrichten-Ebene sowie zur Berücksichtigung der Fehlerbehandlung auf die Übertragungszeit. Abhängig vom Fehlertyp kann bei „Auftreten“ eines Fehlers die Framelänge bestimmt werden.

4.1.4.2 Meta-Modell der Modellbibliothek für CAN

Ebenso wie die Modellkomponenten für FlexRay, lassen sich auch die entwickelten Modellkomponenten für den CAN-Bus in das Meta-Modell einordnen. In Abbildung 4.1.13 ist diese Einordnung basierend auf dem SCS-Meta-Modell dargestellt. Die Basiselemente des Meta-Modells sind in der Grafik eingefärbt.

Für die einzelnen Basiselemente existiert jeweils ein Repräsentant für den CAN. Der *CAN::Cluster* (*Cluster*) bildet eine logische Struktur und gruppiert die einzelnen Knoten zu einem Kommunikationsverbund. Anders als bei FlexRay existieren derzeit keine unterschiedlichen Abstraktionen, so existiert für den *CAN::Channel* und den *CAN::Communication Controller* lediglich mit dem *CAN::Channel_msg* und *CAN::CC_msg* lediglich eine Repräsentation.

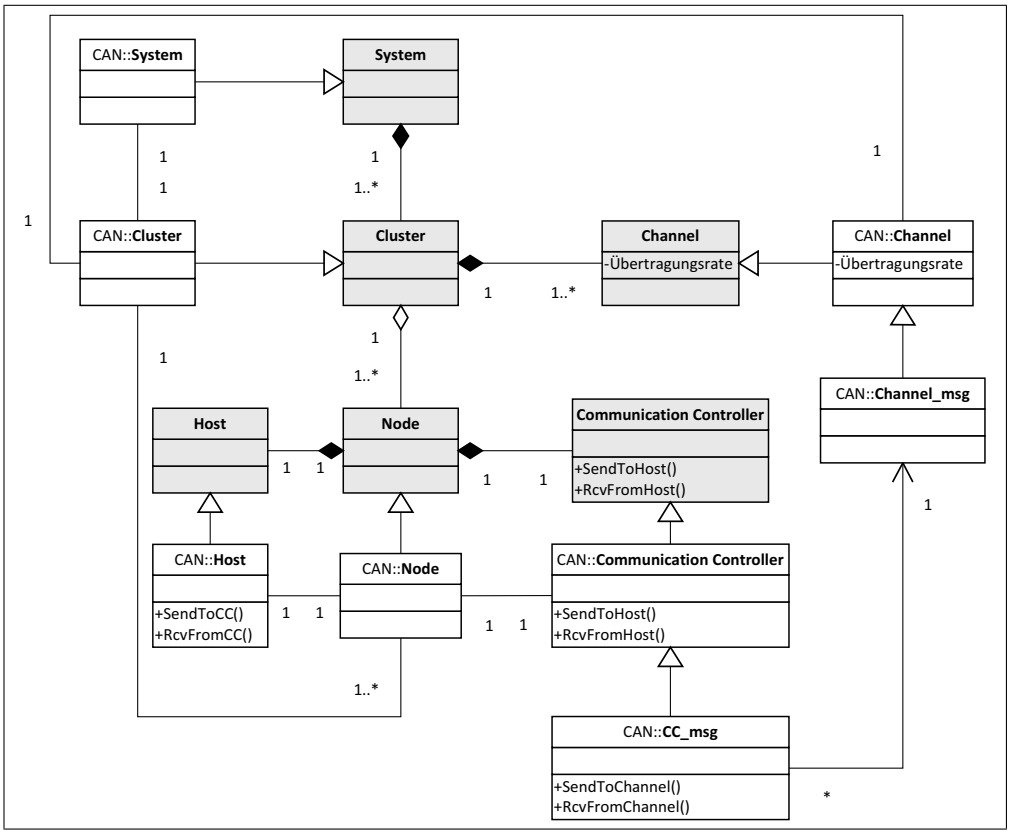


Abbildung 4.1.13: Meta-Modell der Modellbibliothek für den CAN-Bus

4.2 Modelle zur Kopplung von Teilsystemen (Gateway)

4.2.1 Homogene und heterogene Kopplung

Betrachtet man große Systeme, so werden in diesen häufig unterschiedliche Kommunikationssysteme eingesetzt. Zu deren Kopplung werden Gateways eingesetzt. Dieses gilt auch für Fahrzeuge, hier kommen aufgrund unterschiedlichster Anforderungen verschiedene Bussysteme zum Einsatz. Ein zentrales Gateway oder mehrere Gateways verbinden dabei die einzelnen Bussysteme.

Neben dieser heterogenen Verbindung kann auch die Kopplung von gleichen oder gleichartigen Bussystemen sinnvoll sein. Durch Verwendung mehrerer gleichartiger Kommunikationssysteme können die Eigenschaften im Vergleich zu einem einzelnen System positiv beeinflusst werden. So kann durch eine Kopplung die Ausdehnung eines Netzes erhöht werden. Zudem ist eine Steigerung der Anzahl der Teilnehmer ebenso wie die Verbesserung der Leistungsfähigkeit durch eine Partitionierung eines großen Netzes in mehrere gekoppelte kleinere Netze möglich. Hierdurch wird eine Reduzierung der Gesamtkommunikation und eine verbesserte Auslastung der einzelnen Netze erzielt. Eine weitere Möglichkeit ist die redundante Auslegung von Teilnetzen.

Als ein Beispiel für solch eine homogene Kopplung im Automobilbereich kann die Verwendung von mehreren CAN-Bussen gesehen werden, durch diese wird eine Ver-

teilung der Kommunikationslast erreicht. In einer klassischen Architektur mit zwei verwendeten CAN-Bussen wird zwischen dem Antriebs-CAN und dem Komfort-CAN unterschieden. Abgesehen von unterschiedlichen Übertragungsraten ist diese Kopplung homogen. (Wallentowitz & Reif, 2006)

Bei einer homogenen Kopplung wird prinzipiell davon ausgegangen, dass zwischen den einzelnen Kommunikationsteilnehmern keine Unterschiede existieren. Im Transfer von einem realen System in die Modellebene, steht die homogene Kopplung von Teilnehmern für die Gleichheit der Kommunikationstypen und deren Eigenschaften.

Auf der Ebene der Modelle können sich die einzelnen Komponenten zusätzlich durch ihren Grad der Abstraktion unterscheiden. In den einzelnen Modellen ist eine Homogenität im Allgemeinen nicht gegeben, da für eine Vielzahl von Fragestellungen und Analysen ein einheitlicher Abstraktionslevel der einzelnen Komponenten nicht notwendig ist. Somit ergeben sich hinsichtlich einer Kopplung auf Modellebene zwei Aspekte die beachtet werden müssen, dies ist zum Einen der Kommunikationstyp und zum Anderen die Abstraktionsebenen der Komponenten. Entsprechend ergeben sich vier Fälle die unterschieden werden können:

- homogene Abstraktionsebenen - homogene Kommunikation,
- homogene Abstraktionsebenen - heterogene Kommunikation,
- heterogene Abstraktionsebenen - homogene Kommunikation,
- heterogene Abstraktionsebenen - heterogene Kommunikation.

Bei der Kopplung von Kommunikationssystemen innerhalb eines Systemmodells gibt es diverse Anforderungen die in der Verwendung eines Gateways resultieren. Dies kann sich zum einen aus der Verwendung von unterschiedlichen Kommunikationstypen ergeben, d.h. es werden in einem System unterschiedliche Protokolle eingesetzt. Dies entspricht der klassischen Aufgabe eines Gateway wie diese in Kapitel 2.1.6 dargestellt wurden. Zudem kann die Nutzung eines Gateways zur Kopplung unterschiedlich abstrakter Modelltypen genutzt werden, deren Einsatz kann durch die spezifische Modellierungsaufgabe oder Fragestellung, beispielsweise im Sinne einer Restbussimulation und der Untersuchung bzw. Analyse eines einzelnen oder einiger weniger Kommunikationsteilnehmer, deren Verhalten im Detail untersucht werden soll, motiviert sein.

4.2.1.1 Anforderungen unter dem Gesichtspunkt Heterogenität bzgl. des Kommunikationstyps

Die Anforderungen an ein Gateway aus Sicht der Kopplung von Kommunikationssystemen, welche sich durch die verwendeten Protokolle unterscheiden, wurden im Abschnitt 2.1.6 betrachtet. Zu den Anforderungen zählen eine globale Adressierung von Nachrichten bzw. Teilnehmern, die Filterung und Pufferung von Nachrichten und die Steuerung des Informationsflusses.

4.2.1.2 Anforderungen unter dem Gesichtspunkt Heterogenität bzgl. der Abstraktionsebene

Die Komposition von vorgefertigten Modellkomponenten zur Erstellung von Systemen zur Simulation und Analyse beinhaltet allgemein die Problematik im Hinblick auf

unterschiedliche Abstraktionseigenschaften der einzelnen Komponenten.

Ein typisches Modellierungsbeispiel aus dem Bereich der Kommunikationssysteme kann hier eine Restbussimulation oder die Analyse einer einzelnen Komponenten im Gesamtsystem sein. Für einen Teil des Gesamtsystems könnte die Analyse auf einem detaillierten Niveau notwendig sein. Genau für diesen Teil wird ein bitgenaues Kommunikationsmodell ausgewählt. Innerhalb des verbleibenden Teiles des Gesamtsystems wird lediglich ein Kommunikationsmodell auf Nachrichtenebene eingesetzt.

Die Granularität der Ereignisse ist innerhalb der einzelnen Modellkomponenten verschieden. Bei einem Datenaustausch bzw. Kooperation zwischen den einzelnen Modellen muss eine Vermittlung und Übersetzung von Events erfolgen. Dieses ist im Allgemeinen vergleichbar mit der Kopplung unterschiedlicher Kommunikationsprotokolle. Das Gesamtmodell zerfällt in unterschiedliche Bereiche, diese unterscheiden sich durch ihren Grad der Abstraktion und sind jeweils für sich betrachtet in sich homogen. Die einzelnen Bereiche werden durch Kopplung verbunden. Grundsätzlich sind die ableitbaren Anforderungen, Funktionen und Eigenschaften mit denen eines Gateway im Bereich der Kopplung von unterschiedlichen Kommunikationsprotokollen vergleichbar.

Bei der folgenden Modellierung von Gateway-Eigenschaften wird der Aspekt der Heterogenität ausschließlich unter dem Gesichtspunkt der Kopplung unterschiedlicher Kommunikationsprotokolle berücksichtigt. Es wird eine homogene Abstraktionsebene vorausgesetzt. Der Ansatz und die Basiskomponenten können allerdings transferiert werden und bilden die Grundlage für die Verwendung unterschiedlicher Abstraktionsebenen der Kommunikation in einem Modell.

Die Kopplung von Modellen mit einem unterschiedlichen Grad der Abstraktion könnte mit Hilfe eines Gateways dahingehen gelöst werden, dass ein Teil des Gateways mit dem abstrakten Kommunikationsnetz und der andere Teil mit dem detaillierten Kommunikationsnetz verbunden ist. Das Gateway selbst vermittelt zwischen den beiden Netzen.

4.2.2 Gateway Modellarchitektur

Bei der Modellierung von heterogenen Systemen und der Analyse des Verhaltens des gesamten Systems ist der Datenaustausch über unterschiedliche Kommunikationskanäle mit verschiedenen Protokollen von Bedeutung. Dies ergibt die Anforderungen nach einer Gateway Modell-Komponente mit den Eigenschaften Flexibilität, Wiederverwendbarkeit und Austauschbarkeit zur Verbindung von unterschiedlichen Kommunikationstechnologien. In dem Abschnitt 3.2.4 wurde für die Modellierung eines Gateways eine grundlegende Architektur eingeführt, welche nachfolgend ausführlicher betrachtet wird. In Verbindung mit den vorgestellten Modellen für einfache homogene Systeme ist die Kombination und somit die Darstellung von heterogenen Kommunikationssystemen durch Verwendung von adaptierbaren Gateway-Modellen zur Analyse des Systemverhaltens möglich.

Betrachtet man ein System so setzt es sich wie in Abbildung 4.2.1 beispielhaft dargestellt aus zwei Kommunikationsclustern zusammen, welche durch ein Gateway verbunden sind. Ein Cluster bezeichnet eine Anzahl von miteinander verbundenen Teilnehmern (Knoten, Nodes), welche Daten über ein Netzwerk (im konkreten Fall

ein Bussystem) austauschen. Über das Gateway können Daten zwischen Knoten über die Clustergrenzen hinweg ausgetauscht werden.

In einem komplexen System wie einem Automobil (vgl. Abbildung 2.1.1) können unterschiedliche Cluster existieren, welche mit unterschiedlichen Protokollen arbeiten. Wie in Abbildung 4.2.1 ist ein Gateway mit beiden Clustern verbunden, arbeitet als „normaler“ Knoten in den angebundenen Clustern und kann Daten zwischen den beiden Clustern vermitteln.

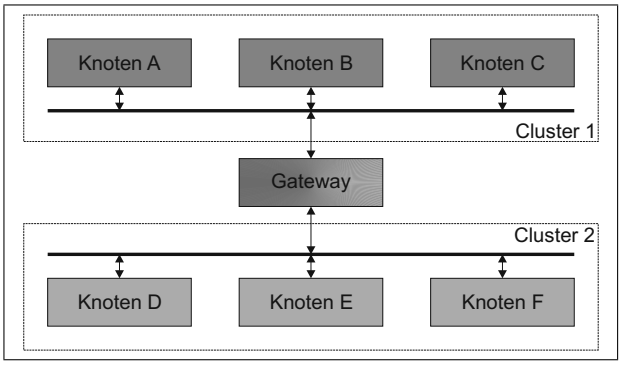


Abbildung 4.2.1: Verbindung zweier Kommunikationscluster durch ein Gateway

Bildet man diese System-Architektur auf eine Modellstruktur ab, analog zum vorgestellten Modellkonzept, so ergibt sich eine Modellstruktur wie sie in Abbildung 4.2.2 zu finden ist. Gleichzeitig wurde eine Verallgemeinerung vorgenommen. Es wird sich im Weiteren nicht auf die Kopplung von zwei Netzen beschränkt, sondern die Verbindung einer „beliebigen“ Anzahl von Netzwerken durch ein Gateway wird unterstützt. Die Modelle sowie das grundlegende Konzept wurden bereits auszugsweise in (Klöckner et al., 2009) veröffentlicht. Die Umsetzung der Modelle erfolgte dabei teilweise in der betreuten Diplomarbeit von (Y. Huang, 2008).

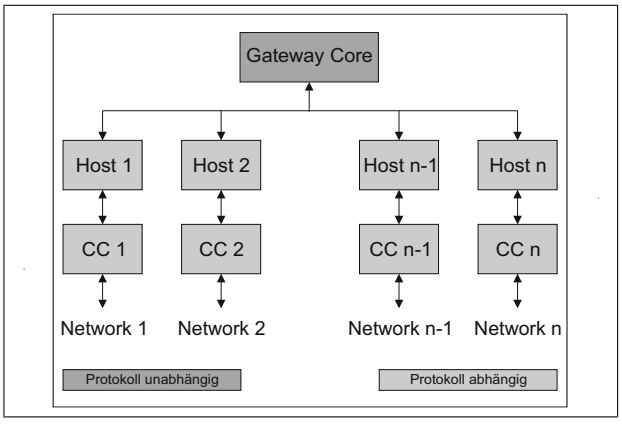


Abbildung 4.2.2: Basis Struktur für ein Gateway-Model

4.2.2.1 Modellbibliothek für die Komponente Gateway

Das Gateway-Modell setzt sich aus zwei Typen von Modulen zusammen: protokoll-abhängige und protokoll-unabhängige Module. Innerhalb der protokoll-unabhängigen Module werden neutrale Gateway Nachrichten und in den protokoll-abhängigen Modulen protokoll-spezifische Nachrichten verarbeitet. Zur Erweiterung von vorhandenen Gateway-Modellen müssen so lediglich die protokoll-abhängigen Module für das jeweilige Kommunikationsprotokoll erstellt werden, somit ist eine Wiederverwendbarkeit und Erweiterbarkeit gewährleistet.

Die Grundstruktur des Gateway Modell in Abbildung 4.2.2 setzt sich aus drei Modultypen zusammen: Gateway Core, Host und CC (Communication Controller).

Das Gateway-Modell basiert auf einem shared medium Ansatz, der zum Austausch von neutralen Nachrichten zwischen Host und Gateway-Core ein internes Netzwerk verwendet.Der mit dem Source-Netzwerk verbundene Host empfängt eine Nachricht über den CC und wandelt diese in eine neutrale Nachricht. Danach wird die Nachricht an den Gateway-Core gesendet. Innerhalb des Gateway-Core wird die neutrale Nachricht verarbeitet und zum Host des Zielnetzwerkes weitergeleitet. Dort wird es in das Format des Zielnetzes umgewandelt. Durch eine eindeutige Kennzeichnung der Hosts ist der Anschluss einer beliebigen Anzahl von Netzwerken möglich.

4.2.2.1.1 Protokoll-unabhängige Komponenten

Gateway Konfiguration Das Modul Gateway kann über ein Interface parametriert werden (zum Beispiel durch Lesen der Konfiguration aus einer Datei). Relevante Parameter zur Konfiguration sind Ziel- und Quell-Netzwerk, der Nachrichten-Identifier im Ziel- und Quell-Netzwerk, die Nachrichtengröße, die Position der Signale innerhalb einer Nachricht sowie ein Zeitlimit für die Weiterleitung. Jedes angeschlossene Netzwerk verfügt über einen Identifier (Port-ID) über diesen werden Quell- und Zielnetzwerk einer Nachricht identifiziert. Abhängig von der Nachrichtengröße ist eine Segmentierung notwendig. Für die Funktionalität eines Signal-Gateway ist die Notation der Position der Signale über die bitgenaue Angabe der Anfangs- und Endposition innerhalb der Nachrichten notwendig. Als Routing Informationen ergeben sich somit folgende Parameter: *Src_PortNr*, *Src_ID*, *Src_Channel*, *FirstBitPos*, *LastBitPos*, *Dest_PortNr*, *Dest_Channel*, *Dest_PayloadLength*, *Dest_ID* und *ExpireTime*. Diese Routing-Informationen können als Modellparameter zu Beginn einer Simulation eingegeben oder über eine externe Konfigurationsdatei eingelesen werden.

Gateway-Core Das zentrale Element des Modells, das Modul Gateway-Core enthält protokoll-unabhängige Funktionen, z.B. Pufferung von empfangenen Nachrichten, Routing von Nachrichten, Segmentierung von großen Nachrichten und Aufspaltung der Nachrichten in Signale (siehe Abbildung 4.2.3).

Funktionsweise Der Gateway-Core empfängt die Nachrichten in dem protokoll-unabhängigen neutralen Format, speichert diese zwischen und segmentiert diese gegebenenfalls. Falls eingehende Nachrichten Signale enthalten, werden diese zunächst extra-

hiert, anschließend in isolierte Nachrichten gepackt und zum Ziel-Kommunikationsteilnehmer in das Zielnetzwerk übermittelt. Die Extraktion der Signale und Weiterleitung von Signalen sowie gesamten Nachrichten ist abhängig von der Konfiguration.

Ablauf Steuerung Die Verarbeitung innerhalb des Gateway Cores kann durch externe Signale gesteuert werden. In dem normalem Betriebszustand werden eingehende Nachrichten anhand der Signal-Routing- und PDU-Routing-Tabelle verarbeitet. Basierend auf der Spezifikation von Funktionsbausteinen mit standardisierten Schnittstellen ist eine hohe Anpassungsfähigkeit der Architektur erreicht und ermöglicht damit die Erweiterung und den Austausch einzelner Funktionen. Mögliche Analysen betreffen unter anderem unterschiedliche Pufferstrategien, welche die Leistung des Systems beeinflussen.

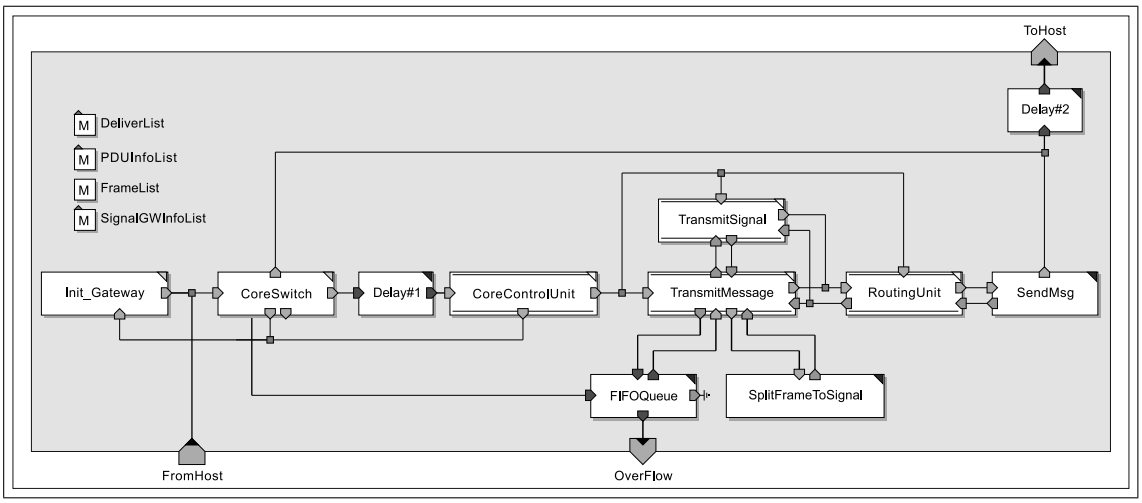


Abbildung 4.2.3: Modell des Gateway-Core

4.2.2.1.2 Protokoll-abhängige Komponenten Die protokoll-abhängigen Funktionen werden innerhalb der jeweiligen Host-Module realisiert (siehe Abbildung 4.2.4). Hierzu zählen beispielsweise Protokolltransformation (protokoll-unabhängige neutrale Nachrichten in protokoll-abhängige Nachrichten), Initialisierung des Kommunikationscontrollers, Nachrichtenübertragung und Nachrichtenempfang.

Der Communication Controller (CC) ist direkt mit dem Netzwerk verbunden und realisiert das Kommunikationsprotokoll inklusive Buszugriffsverfahren, Fehlererkennung und Synchronisation. Entsprechend der service-orientierten Modellierung bietet der CC dem Host verschiedene Dienste an. Die Elemente CC und Bus entsprechen dabei Modellen wie sie in den Kapiteln 4.1.3.1 und 4.1.4.1 für die Kommunikationssysteme vorgestellt wurden.

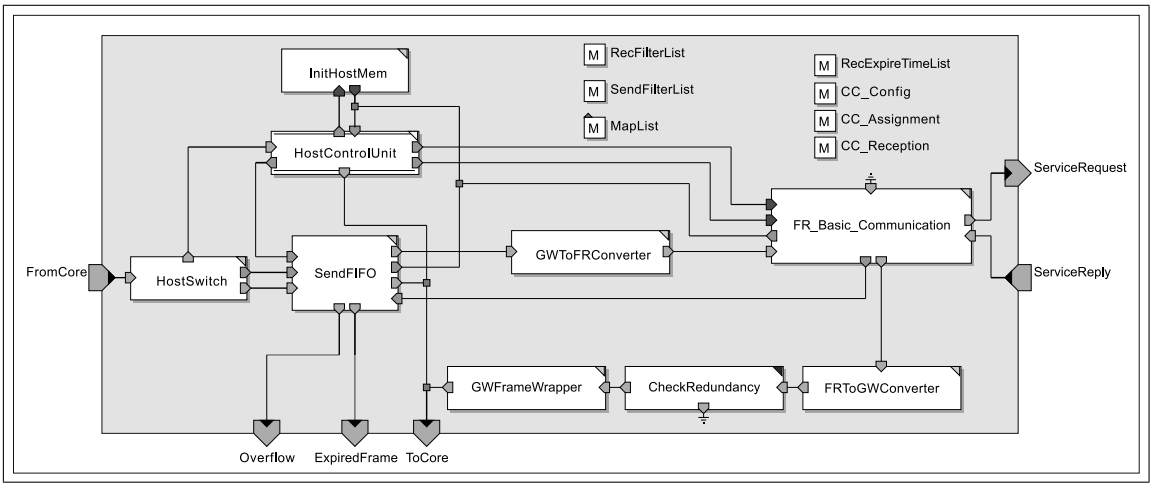


Abbildung 4.2.4: Modell des FlexRay-Host zur Anbindung des FlexRay an das Gateway

4.2.2.2 Meta-Modell der Modellbibliothek für ein Gateway

Wie zuvor werden die erstellten Modellkomponenten für das Gateway in das Meta-Modell eingeordnet. Die Einordnung der Gateway-Komponenten erfolgt auf Grundlage des HCS-Meta-Modells, welches heterogene Architekturen berücksichtigt. Abbildung 4.2.5 zeigt einen Ausschnitt des Meta-Modells inklusive der Modellkomponenten für ein Gateway. Die Basiselemente des Meta-Modells sind in der Grafik eingefärbt.

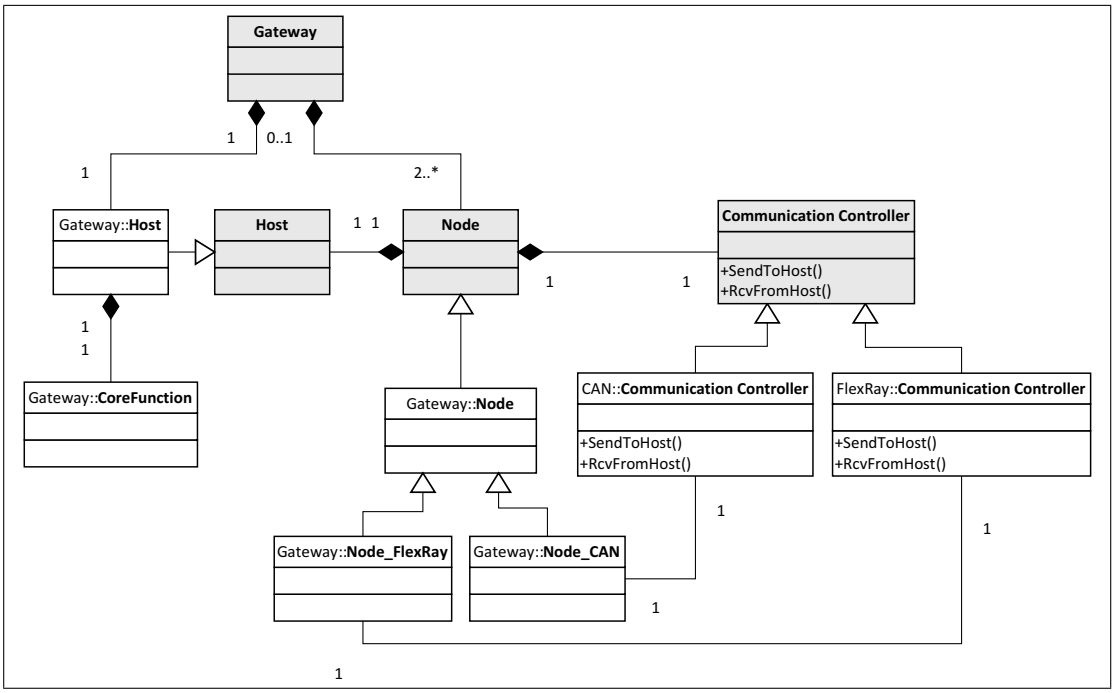


Abbildung 4.2.5: Meta-Modell der Modellbibliothek für ein Gateway

Die beiden Elemente *CAN::Communication Controller* und *FlexRay::Communica-*

tion Controller sind bereits aus den Meta-Modellen für FlexRay (Abbildung 4.1.9) und CAN (Abbildung 4.1.13) bekannt.

Ein Gateway setzt sich zusammen aus einem spezifischen Host-Element, dem protokoll-unabhängigen *Gateway::Host*, dieser beinhaltet und realisiert die Basisfunktionen eines Gateway, sowie mehrere Node-Elemente (*Gateway::Node*), welche an dieser Stelle für die Protokolle CAN (*Gateway::Node_CAN*) und FlexRay (*Gateway::Node_FlexRay*) vorhanden sind. Diese protokollspezifischen Elemente vermitteln zwischen den zugehörigen Communication Controllern und den Basisfunktionen des Gateway, d.h. sie übernehmen die Transformation zwischen den protokoll-abhängigen Datenformaten und dem protokoll-unabhängigen Datenformat.

4.2.2.3 Anwendungsbeispiel: Gateway FlexRay-CAN

In der betreuten Diplomarbeit von Y. Huang (2008) wurden die erstellten Modellkomponenten durch den Vergleich einer Simulation mit einer Verhaltensspezifikation getestet. Zum Nachweis der Korrektheit des entwickelten Modells wurde das nachfolgend beschriebene Referenzsystem und verschiedene Testszenarien spezifiziert. Durch Analyse der Simulationsergebnisse konnte das korrekte Verhalten der erstellten Modelle nachgewiesen werden.

Die Anwendung des entwickelten Modellkonzeptes für Gateways wird an der Modellkomponente eines FlexRay-CAN-Gateway demonstriert und die Grundstruktur und die Zusammenhänge der Gateway Modell-Komponenten veranschaulicht.

Das in Abbildung 4.2.6 dargestellte System enthält zwei FlexRay- und zwei CAN-Netzwerke, die über ein Gateway verbunden sind. Jedes Netzwerk enthält zwei oder mehr Knoten. Das System selbst ist eine Erweiterung einer Anwendung dargestellt in (Hedenetz & Belschner, 1998, S.2).

Die Komponente Gateway besteht aus dem Modul Gateway Core sowie den Modulen Host und CC (Communication-Controller) der angebundenen Bussysteme (vgl. Abbildung 4.2.2). Der Austausch von Daten zwischen den Netzwerken wird durch den „virtuellen Bus“ zwischen Gateway Core und den jeweiligen Host Modulen (zwei FR_GW_Host-Module für die Anbindung von FlexRay und zwei CAN_GW_Host-Module für die Anbindung von CAN) realisiert (vgl. Kapitel 4.1.3.1 und 4.1.4.1). Analysiert werden können u.a. die Einhaltung der Echtzeit Anforderungen von Nachrichten (Eigenschaften Overflow oder ExpiredFrame)

In jedem der über ein Gateway verbundenen vier Cluster befinden sich zwei oder mehrere Teilnehmer (Node). Die beiden FlexRay-Cluster sind wie in Tabelle 4.4 konfiguriert. Die Zuordnung der Slots zu den einzelnen Teilnehmern ist in Tabelle 4.5 zu finden.

Innerhalb des CAN-Cluster werden das Nachrichtenformat „Standard“ und die Datenrate von 500 $kbit/s$ eingestellt. Die einzelnen Idendifier der Nachrichten und ihre Größe (Payload) sind der Tabelle 4.6 entnehmbar.

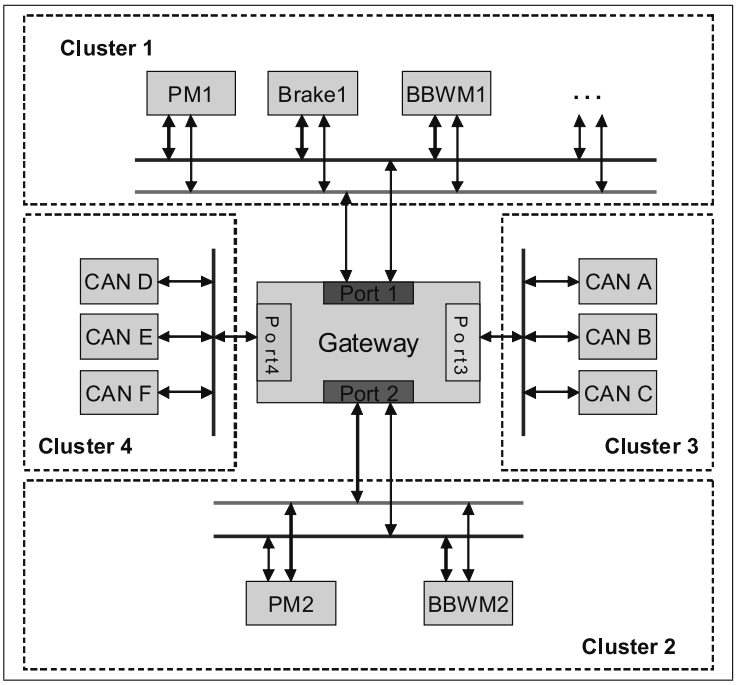


Abbildung 4.2.6: Darstellung des Systems zur Validierung des Modells

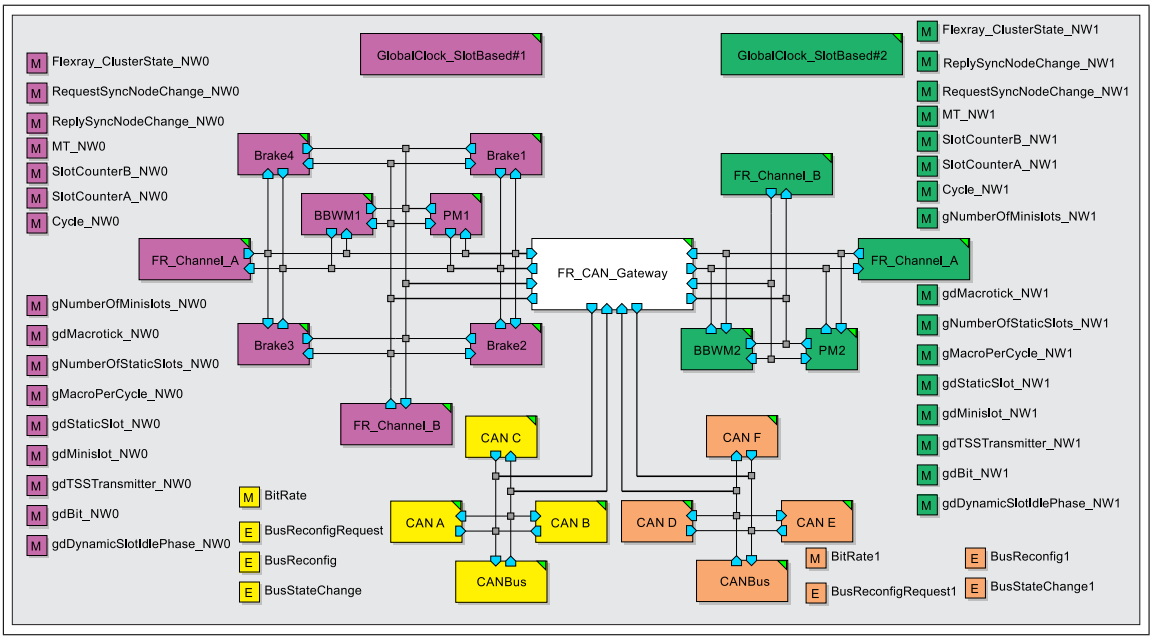


Abbildung 4.2.7: Beispiel für ein Modell eines Gesamtsystems mit einem Gateway und jeweils zwei CAN- und FlexRay-Systemen

Abschließend wird das Gateway für ein Routing von Signalen wie in Tabelle 4.7 angegeben konfiguriert.

Tabelle 4.4: Konfigurationsparameter der FlexRay-Cluster

Datenrate	$10 Mbit/s$
Anzahl Statischer Slots	16
Macrotick pro Statischem Slot	100
Anzahl Minislots im Dynamischen Segment	170
Übertragene Daten im Statischen Slot	240 $Byte$
Dauer eines Macrotick	5 μs
Dauer eines Zyklus	10 ms

Tabelle 4.5: Konfigurationsparameter der Slots in den einzelnen FlexRay-Cluster

Netzwerk	Node	Slot ID
FR_1	PM1	2, 10
FR_1	Brake1	5, 13
FR_1	Brake2	6, 14
FR_1	Brake3	7, 15
FR_1	Brake4	8, 16
FR_1	BBWM1	1, 9
FR_1	Gateway	33, 34, 35, 36
FR_2	PM2	4, 12
FR_2	BBWM2	3, 11
FR_2	Gateway	33, 34, 35, 36

Tabelle 4.6: Konfigurationsparameter der einzelnen CAN-Cluster

Netzwerk	Node	Slot ID	Payload
CAN_1	CAN A	4	1 Byte
CAN_1	CAN B	5	3 Byte
CAN_1	CAN C	1	8 Byte
CAN_1	Gateway	2	4 Byte
CAN_2	CAN D	7	8 Byte
CAN_2	CAN E	8	3 Byte
CAN_2	CAN F	1	1 Byte
CAN_2	Gateway	9	2 Byte

Tabelle 4.7: Konfigurationsparameter des Gateway Routing

Netzwerk (Quelle)	Nachricht (Quelle)	Netzwerk (Ziel)	Nachricht (Ziel)
Cluster 1	ID 10	Cluster 1, Cluster 2	ID 34
Cluster 1	ID 10	Cluster 3	ID 2
Cluster 1	ID 10	Cluster 1, Cluster 2	ID 35
Cluster 3	ID 4	Cluster 1, Cluster 2	ID 33
Cluster 2	ID 11	Cluster 4	ID 9

4.3 Modelle zur Berücksichtigung von komplexen Zusammenhängen auf Anwendungsebene: Betriebssystemfunktionalität (Host)

Bei der Betrachtung des Verhaltens eines Systems, insbesondere bezüglich des Zeitverhaltens, ist neben der Kommunikation auch die übergeordnete Schicht relevant. In komplexen verteilten Systemen kommen in den einzelnen Geräten zumeist Betriebssysteme zum Einsatz, die durch ihre Mechanismen und zeitlichen Eigenschaften einen Einfluss auf das Systemverhalten nehmen können. In Abschnitt 3.2.4 wurde die Integration dieser in den grundsätzlichen Modellaufbau vorgestellt. Darauf aufbauend werden als ergänzendes Beispiel zur Modellierung der Kommunikation nachfolgend exemplarisch zwei Beispiele kurz aufgeführt. Dies sind die in Abschnitt 2.2 vorgestellten Betriebssystemstandards aus dem Automotive Segment OSEK/OS und OSEKtime/OS. Die prinzipielle Erweiterbarkeit der Modellkomponente Host wird demonstriert. Die praktische Umsetzung der Modelle erfolgte in der betreuten Diplomarbeit von Kappert (2008).

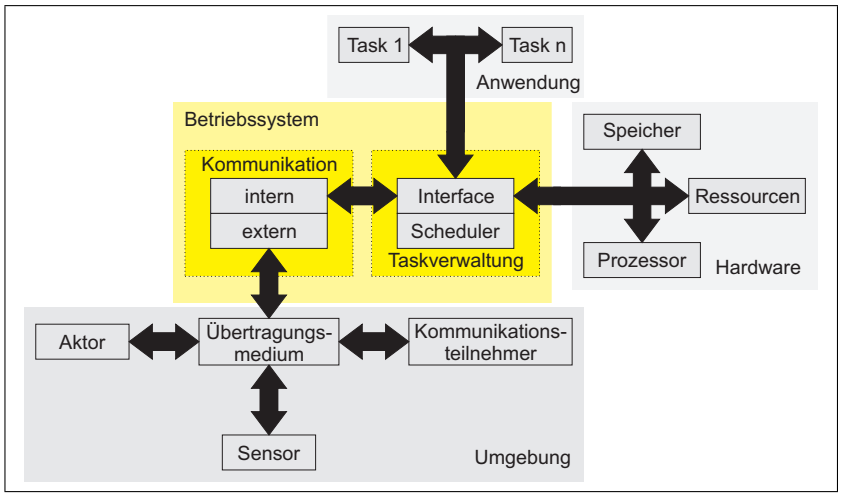


Abbildung 4.3.1: Allgemeine Struktur eines Knotens

Betrachtet man nun den vereinfachten Aufbau eines einzelnen Netzknoten und seine Umgebung, so ist es möglich diesen wie in Abbildung 4.3.1 dargestellt in vier Teilbereiche aufzuteilen: die Anwendung welche durch verschiedene Tasks realisiert bzw. repräsentiert wird; das Betriebssystem in einer einfachen Form inklusive der Funktionen Taskverwaltung, Kommunikationssteuerung und Steuerung des Hardwarezugriffs; die Hardware mit Prozessor, Speicher sowie weiteren Ressourcen; die Umgebung mit Aktoren, Sensoren und externen Kommunikationsteilnehmern. Eingegliedert in das Modellierungskonzept ergibt sich die in Abbildung 4.3.2 dargestellte Modellarchitektur. Die Modellierung der Applikationsebene (Tasks) sowie die Berücksichtigung von konkreten Hardwarekomponenten und Ressourcen eines Knotens werden im Folgenden nicht weiter betrachtet. Das Betriebssystem inklusive der Anwendung und der Hardware lassen sich auf Grund ihrer Eigenschaften der Modellkomponente Host zu-

ordnen. Die Anbindung an die Umgebung und der Zugriff auf das Übertragungsmedium werden über die Modellkomponenten Channel und Communication Controller realisiert.

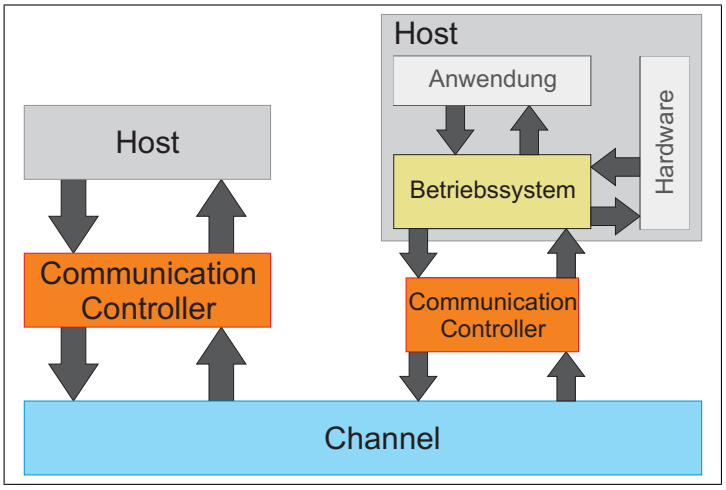


Abbildung 4.3.2: Einordnung der Knotenstruktur in die Grundstruktur des entwickelten Modellierungsansatzes

Das Betriebssystem untergliedert sich vereinfacht in Systemkern, Kommunikation und Scheduler und hat dabei als Hauptaufgabe die Steuerung der Task-Ausführung und die damit verbundene Ressourcenzuteilung und Prozessorzuweisung. Eine ausführliche Betrachtung von Betriebssystemen, deren allgemeinen Aufgaben und Funktionen finden sich ausführlich in (Tanenbaum, 2009) und (Baumgarten & Siegert, 2007). Die Ausführung eines Task ist abhängig vom Schedulingverfahren und den Eigenschaften (z.B. Laufzeit, Ressourcen, Priorität). Die Kommunikation erfolgt auf Basis Interprozesskommunikation sowie über spezifische Kommunikationsprotokolle (CAN, FlexRay u.ä.).

Die nachfolgend vorgestellte grundlegende Struktur für die Modellierung der Betriebssystemstandards OSEK/OS und OSEKtime/OS (Abschnitt 2.2) folgt der strukturellen Einteilung in Betriebssystem, Anwendung, Hardware und Umgebung. Die realisierten Modelle geben ein Beispiel für die Umsetzung von Simulationsmodellen für konkrete Betriebssysteme in realen Systemen.

4.3.1 Allgemeine Struktur der Modell-Komponente Betriebssystem

In der grundlegenden Modellstruktur für ein Betriebssystem werden die vier Hauptkomponenten berücksichtigt: Task, Scheduler, Kommunikation und Ressourcen. Diese allgemeine Struktur erlaubt einen einfachen Modulwechsel sowie die Anpassung an unterschiedliche Betriebssysteme. Den Hauptkomponenten werden dabei folgende Funktionen zugeordnet.

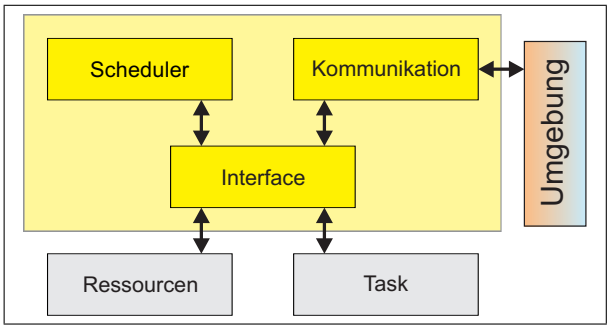


Abbildung 4.3.3: Darstellung der Basis Modellkomponenten eines Betriebssystems

Task Ein Task ist ein eindeutig identifizierbarer, einzelner Prozess einer Anwendung mit einer gewissen Anzahl an möglichen Zuständen. Die Zustände sind dabei ebenso wie der Zugriff auf Betriebssystemfunktionen abhängig vom konkreten Betriebssystem. Im Allgemeinen sind mindestens drei Zustände oder eine Variante dieser vorhanden: aktiv, verdrängt und beendet.

Jeder Task hat eine Laufzeit, welche die maximale Zeit im aktiven Zustand angibt, und in Echtzeitbetriebssystemen eine Deadline, diese kann hart oder weich sein.

Scheduler Die Abarbeitung der Tasks innerhalb des Betriebssystems wird durch den Scheduler gesteuert, die Steuerung erfolgt nach betriebssystemspezifischen Kriterien.

Kommunikation Die Interprozesskommunikation ermöglicht den Austausch von Daten zwischen Tasks über Nachrichten, Shared Memory oder Events. Ergänzend zu der internen muss die externe Kommunikation über Protokolle wie CAN und FlexRay unterstützt werden.

Ressource Die verfügbaren Ressourcen wie Prozessor und Speicher werden vom Betriebssystem verwaltet. Mit geeigneten Synchronisationsmechanismen wird der Zugriff koordiniert und synchronisiert. Jeglicher Ressourcenzugriff erfordert den Betriebssystemeingriff dazu zählen u.a. Taskaktivierung und Speicherverteilung.

4.3.2 Beispiel OSEK/OS und OSEKtime/OS

4.3.2.1 Anforderungen OSEK/OS

Zunächst werden noch einmal kurz die Eigenschaften von OSEK/OS im Hinblick auf die Modellkomponenten aufgegriffen.

Task Ein OSEK/OS-Task besitzt eine Priorität und kann vier Zustände (running, waiting, suspended und ready) annehmen, der Startzeitpunkt eines Tasks ist beliebig. Auf Grundlage der Prioritäten können Tasks verdrängt werden. Die Eigenschaft der Verdrängbarkeit eines Tasks ist konfigurierbar. Zudem ist Tasks das Erzeugen und Konsumieren von Events, der Start anderer Tasks oder der

Selbststart sowie das Reservieren und Freigeben von Ressourcen erlaubt. Die Verwaltung der Tasks erfolgt durch den Scheduler.

Scheduler Der Scheduler arbeitet prioritätsgesteuert, somit wird jeweils der höchstpriore Task ausgewählt und gestartet. Bei der Aktivierung werden u.a. die maximale Aktivierungszahl und die Einstellung der Verdrängbarkeit geprüft.

Kommunikation Jeder Task ist fähig Nachrichten zu verschicken und unterscheidet dabei nicht zwischen interner und externer Kommunikation. Die Interprozesskommunikation zwischen Tasks der Systeme OSEKtime/OS und OSEK/OS wird über „unqueued messages“ realisiert. Zudem können OSEK/OS-Tasks über Events kommunizieren.

Ressource Aufgrund der Prioritätssteuerung und des beliebigen Startzeitpunktes eines Tasks wird das Priority Ceiling Protokoll zur Synchronisation des Zugriffes auf Ressourcen verwendet. Hierbei wird jeweils einer Ressource eine Priorität zugewiesen. Diese muss höher sein, als die eines jeden zugreifenden Tasks. Bei Ressourcenzugriff erhält der Task die Priorität der Ressource, solange er diese nutzt. Somit kann kein weiterer Task auf diese Ressource zugreifen.

4.3.2.2 Anforderungen OSEKtime/OS

Task Ein Task in OSEKtime/OS (ttTask) besitzt drei Zustände: running, preempted und suspended. Die Startzeit eines ttTask ist fest bei variabler Laufzeit. Ein ausgezeichneter Task (ttIdleTask) ist aktiv, sobald kein anderer ttTask aktiv ist. Daher besitzt er lediglich die zwei Zustände running und preempted.

Scheduler Der Scheduler in OSEKtime/OS wird als Dispatcher bezeichnet und aktiviert anhand einer Tabelle die einzelnen Tasks. Diese Dispatcher-Tabelle enthält Einträge wie TaskID, Startzeitpunkt und Deadline. Der ttIdleTask findet sich nicht in der Tabelle und wird zu Abarbeitungsbeginn gestartet und nicht beendet. Wird entsprechend der aktuellen Zeit ein Task aktiviert, so wird der bisherige Task verdrängt und anschließend wieder gestartet (Stackmechanismus).

Ressource Ein spezifischer Mechanismus zur Steuerung des Ressourcenzugriffes existiert nicht, da dieser a-priori zum Zeitpunkt der Systemgenerierung bekannt ist.

4.3.2.3 Modelle

4.3.2.3.1 Taskmodell (Anwendungsebene) Für die Modellierung von Tasks gibt es unterschiedliche Möglichkeiten. Dabei werden der Zustand und die Eigenschaften des Tasks in verschiedenen Varianten verarbeitet und verwaltet. Hierfür können innerhalb von MLDesigner insbesondere die Elemente Modul, Memory sowie Event genutzt werden, wobei der Task-Zustand und die Eigenschaften jeweils durch eine Datenstruktur repräsentiert werden. Die Vor- und Nachteile einzelner Varianten sollen an dieser Stelle nicht vertieft werden.

In dieser Modellvariante wird ein Task als einzelnes Modul realisiert, welches die Funktionen eines Tasks innerhalb des jeweiligen Betriebssystems repräsentieren. Eine Anwendung setzt sich aus unterschiedlichen Tasks zusammen. Somit ist in der Modellumgebung eine sehr einfache intuitive Anwendungserstellung auf Basis der Task-Module möglich. Diese einfache Komposition ist der große Vorteil der Modul-Variante und wird daher gewählt. Durch die hierarchische Strukturierung und Gliederung sind einfache Substitutionen möglich und entsprechen der allgemeinen Austauschbarkeit unterschiedlicher Komponenten im Sinne der Abstraktion.

OSEK/OS Task Der Zustand und die Eigenschaften eines OSEK/OS-Task werden in einer Datenstruktur gehalten über welche auch die Signalisierung erfolgt. Dies entspricht einem Stack auf dem ein Task seine Daten bei Verdrängung speichert. Die Datenstruktur enthält folgende Elemente: *TaskID* - zur eindeutigen Identifizierung eines Tasks, *State* - zur Darstellung des Zustand eines Tasks, *Priority* - zur Festlegung der Priorität des Tasks, *RunTime* - gibt die Laufzeit des Tasks an und *ActualTime* - Speicherung der verstrichenen Bearbeitungszeit bei Verdrängung.

Zusätzlich gibt es noch den Kennzeichner OSEKtime_preempted, der angibt ob ein Task von OSEKtime verdrängt worden ist, da ein OSEK/OS als Subsystem eines OSEKtime/OS vorhanden sein kann.

Das Verhalten eines Tasks lässt sich über die folgenden Parameter konfigurieren: *RunTime* - Festlegung der durchschnittlichen Tasklaufzeit; *Priority* - Festlegung der Taskpriorität; *Preemptiv* - Einstellung der Verdrängbarkeit des Tasks; *SendMessageID* - Identifier der zu sendenden Nachrichten; *MessageSendingTime* - Zeitpunkt während der Abarbeitung eines Tasks zu dem eine zugeordnete Nachricht gesendet wird; *ReceiveMessageID* - Identifier der zu empfangenden Nachrichten; *Extended* - Unterscheidung des Taskmodus Basic oder Extended; *InternalDelay* - durch Taskübergänge entstehende Verzögerungszeit; *ReceiveEvent* - Events auf die im Zustand Waiting gewartet wird; *Event_Send_Times* - Zeitpunkte zu denen Events gesendet werden; *Cyclic_Activation* - Zeitlicher Abstand zwischen zwei Selbstaktivierungen; *Task_Activation* - Identifier und Priorität eines Tasks der aktiviert wird; *Ressource_Usage* - Identifier der Ressource sowie Zeitpunkt der Anforderung und Ressourcenhaltezeit.

OSEKtime/OS Task Der Zustand und die Eigenschaften eines ttTask werden ebenso wie bei dem OSEK/OS-Task in einer Datenstruktur gehalten über welche auch die Signalisierung erfolgt. Die Datenstruktur enthält die folgenden Elemente: *TaskID* - zur eindeutigen Identifizierung eines Tasks, *ttTaskStateType* - zur Darstellung des Zustand eines Tasks, *Deadline* - zur Festlegung des Zeitpunktes an dem der Task beendet sein muss, *RunTime* - gibt die Laufzeit des Tasks an, *StartTime* - zur Sicherung des Zeitpunktes der Task-Erstaktivierung und *ActualTime* - zur Speicherung der verstrichenen Bearbeitungszeit bei Verdrängung.

Der ttIdleTask hat standardmäßig den Identifier Null. Das Verhalten eines Tasks lässt sich durch folgende Parameter konfigurieren: *minTime, maxTime* - Fest-

legung der minimalen und maximalen Ausführungszeit eines Tasks zur Bestimmung der RunTime, *Deadline* - Zeitpunkt innerhalb einer Dispatcher Round zu dem der Task beendet sein muss, *InternalDelay* - durch Taskübergänge entstehende Verzögerungszeit, *SendMessageID* - Identifier der zu sendenden Nachrichten, *MessageSendingTime* - Zeitpunkt während der Abarbeitung eines Tasks zu dem eine zugeordnete Nachricht gesendet wird und *ReceiveMessageID* - Identifier der zu empfangenden Nachrichten.

4.3.2.3.2 Scheduler

OSEKtime/OS Dispatcher Zu den wichtigen Systemparametern für den Dispatcher zählen die Zeitdauer der Dispatcher Runde (RoundTime) sowie die Tasks mit ihren Startzeiten (Dispatchertabelle). Die Taskverwaltung erfolgt auf Basis der Task-Datenstrukturen. Die Liste der Tasks wird innerhalb einer Dispatcher Runde abgearbeitet und anschließend wird die Taskaktivierung erneut gestartet. Zudem erfolgt die Verwaltung verdrängter Tasks. Ein solcher wird vom Stack genommen sobald ein Task in den Zustand Suspended wechselt. Zuunterst im Stack befindet sich der ttIdle-Task.

Innerhalb des Moduls ist durch Substitution die Verwendung anderer Schedulingverfahren möglich. Dabei stehen Modelle für folgende Verfahren zur Verfügung: ein OSEKtime kompatibler Stack-Algorithmus, *highest ActualTime* Algorithmus, *highest RemainingTime* Algorithmus, *highest RunningTime* Algorithmus, *longest Deadline* Algorithmus, *lowest ActualTime* Algorithmus, *lowest RemainingTime* Algorithmus, *lowest RunningTime* Algorithmus und *shortest Deadline* Algorithmus.

OSEK/OS Scheduler Der Scheduler verwaltet die Tasks des OSEK/OS-Systems. Ein wichtiger Parameter ist die Rescheduling_Time, welche den Abstand zwischen zwei Rescheduling-Vorgängen angibt. Bei einem Rescheduling-Vorgang wird der höchstpriorisierte Task ausgewählt.

Bei der Aktivierung eines Tasks erfolgt die Prüfung der maximalen Aktivierungsanzahl, diese wird bei Aktivierung erhöht und bei Beenden eines Tasks reduziert. Die Anzahl an aktiven Tasks wird dadurch beschränkt.

Ergänzend erfolgt das Aussenden von Events zur Reaktivierung von wartenden Tasks.

OSEK/OS als Subsystem von OSEKtime/OS Ein OSEKtime/OS kann als Subsystem ein „normales“ OSEK/OS integrieren. Daher sind die Modell-Teile so konzipiert, dass sie miteinander kombiniert werden können. Der Scheduler ist somit eine Kombination aus Dispatcher und Scheduler, der abhängig von der Konfiguration und den Tasks arbeitet. Der OSEK/OS Scheduler verdrängt den ttIdleTask und übernimmt dessen Aufgabe.

Diese Kopplung wird an dieser Stelle als obligatorisch angesehen und nachfolgend die Kommunikation für beide Systeme gemeinsam dargestellt. Es ist offensichtlich, dass durch geeignete Konfiguration beide Systeme auch getrennt laufen können.

4.3.2.3.3 Kommunikation Das OSCOM Modul ist für die interne und externe Kommunikation von OSEKtime und OSEK OS Tasks zuständig. Die Kommunikation ist ausschließlich über Nachrichten möglich, dabei dient der Instruction Layer des OSEK OS als Basis und wird um eine fehlertolerante Kommunikation (FTCOM) ergänzt. Für die Kommunikation zwischen den Tasks und diesem Modul werden die Datenstrukturen sendMessage (MsgID, Data), receiveMessage (MsgID, Task ID) und sendData (TaskID, Data) definiert.

Aus Kompatibilitätsgründen zwischen beiden Systemen werden „unqueued“ Messages unterstützt, Nachrichten ändern sich durch überschreiben und können jederzeit gelesen werden.

Zentrale Aufgaben sind das Routing von Nachrichten und die Nachrichtenfilterung. Bei dem Weiterleiten von Nachrichten wird zwischen interner und externer Kommunikation unterschieden. Zur Realisierung einer externen Kommunikation werden interne in externe Nachrichtenformate und vice versa umgewandelt. Eine entsprechende Zuordnung der Nachrichten ist konfigurierbar. Die externe Kommunikation wurde exemplarisch in Verbindung mit CAN-Nachrichten untersucht.

4.3.2.3.4 Umgebung Das Umgebungsmodell ist nicht Teil der Betriebssystem-Modellbibliothek und repräsentiert die Umgebung eines einzelnen Knotens. Dies entspricht einem Kommunikationssystem mit mehreren Knoten, die über ein Kommunikationsmedium miteinander kommunizieren. Das Umgebungsmodell enthält beispielsweise die zuvor vorgestellten Komponenten aus den Bibliotheken CAN und FlexRay. Durch die selektive Modellierung eines Betriebssystems in den einzelnen Knoten, lassen sich auf Ebene der Anwendung unterschiedliche Abstraktionen erreichen.

4.3.2.3.5 Ressourcen Tasks innerhalb eines OSEK OS Systems können auf bis zu acht Ressourcen zugreifen. Jeder Ressource sind ein eindeutiger Bezeichner und eine Priorität zugeordnet. Der Zugriff und das Freigeben erfolgt über die Dienste GetRessource, SetPriority und ReleaseRessource. Zwei Tasks können aufgrund des realisierten Priority Ceiling Protocol nicht gleichzeitig auf eine Ressource zugreifen. Beliebige Ressourcen lassen sich bei Modellierung eines konkreten Betriebssystems integrieren.

4.4 Automatische FIBEX-basierte Modellgenerierung

Die komfortable Konfiguration und Beschreibung von Kommunikationsstrukturen kann über FIBEX und zugehörige Tools geschehen. Der Standard FIBEX erlaubt die umfangreiche Spezifikation eines Gesamtsystems und unterstützt u.a. die Bussysteme CAN, LIN, MOST und FlexRay. Eine einfache Kopplung zwischen FIBEX und MLDesigner ist durch eine geeignete Modelltransformation möglich. Somit ist die Parametrisierung und einfache Generierung der FlexRay- und CAN-Modelle auf Basis einer FIBEX-Beschreibung möglich. Weiterführende Informationen zu der Transformation finden sich im Anhang A.5.

4.4.1 FIBEX2MLD-Transformation

Die Transformation der Daten aus dem FIBEX-Format in das ebenfalls XML-basierte MLDesigner Systembeschreibungsformat erfolgt über eine XSL-Transformation, welche eine einfache Transformation von XML-Daten erlaubt (Vonhoegen, 2005).

Die Regeln für die Transformation von FIBEX-Elementen in korrespondierende Modell-Komponenten (diese entsprechen Modulen in MLDesigner) sind in einem XSLT-Skript hinterlegt. Berücksichtigt werden Cluster, Channels sowie alle Controller und ECUs. Die Abbildungsvorschrift wird dabei in einer Datei (MAP-Datei) auf XML-Basis definiert, hier werden die einzelnen Elemente von FIBEX auf MLDesigner Modellelemente abgebildet. Für die Bussysteme existieren für die unterschiedlichen Abstraktionsgrade jeweils ein XSLT-Skript und eine MAP-Datei.

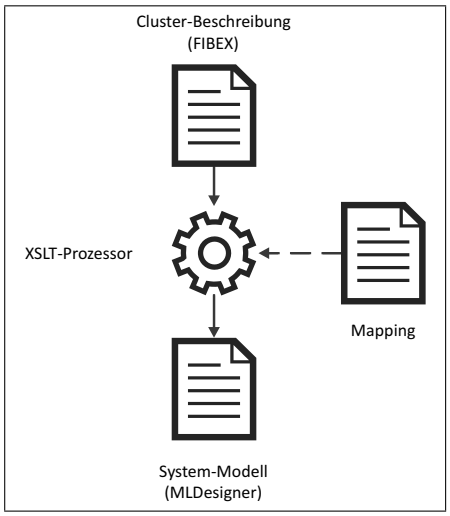


Abbildung 4.4.1: Ablauf der Transformation

Somit ergibt sich für die Transformation der in Abbildung 4.4.1 dargestellte Ablauf. Aus einer vorliegenden Systembeschreibung in einem FIBEX-Format wird eine Transformation auf Basis eines ausgewählten XSLT-Skriptes und einer spezifischen MAP-Datei zur Generierung eines System-Modells ausgeführt. Das generierte System-Modell kann anschließend in MLDesigner geöffnet, bearbeitet und simuliert werden.

4.4.2 Anwendungsbeispiel

Die prototypische Implementierung des Transformationsprozesses erfolgt auf Basis der FIBEX Version 2.01 (ASAM, 2007). Das XSLT-Stylesheet verarbeitet ein FIBEX-

Dokument, welches eine CAN-Bus-Systembeschreibung enthält und überführt es in das Dateiformat des MLDesigner-Programms. Dazu werden MLD-Klassendateien verwendet, welche die entsprechenden Systemelemente bereitstellen. Durch die Map-Datei werden die Zusammenhänge der CAN-Elemente des FIBEX und den korrespondierenden MLD-Klassen definiert, sowie zusätzliche Information für das Zielformat bereitgestellt.

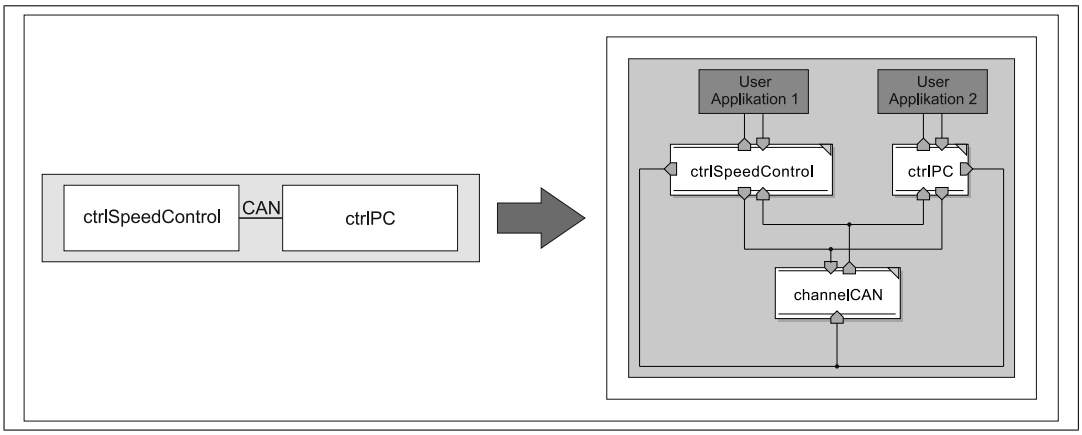


Abbildung 4.4.2: Beispiel zur automatischen Generierung von Modellen

Die FIBEX-Beschreibung enthält ein einfaches System bestehend aus zwei Knoten die über einen CAN-Bus verbunden sind. Diese Beschreibung ist das Eingangsdokument für die Generierung des Modells und ist ein begleitendes Beispiel zur Spezifikation des Standards. Die Abbildung 4.4.2 stellt das Eingangssystem dar und zeigt zudem das erzeugte MLDesigner Modell. Die betrachtete Modellbibliothek umfasst keine Applikationsmodelle. In dem MLDesigner-Modell sind daher die applikationsspezifischen Komponenten gekennzeichnet. Deutlich zu erkennen sind die beiden Kommunikationscontroller und das Bus-Element.

Durch diese Verbindung der Modellierungs- und Simulationsebene und der XML-basierten Spezifikation und Beschreibung von Kommunikationsclustern wird die Untersuchung und Analyse von Systemen erleichtert.

4.5 Zusammenfassende Übersicht der entstandenen Bibliotheken für die Modellierung

Im Modellierungstool MLDesigner sind einige Bibliotheken für die Modellierung von Systemen entstanden, abschließend sollen die wesentlichen zusammenfassend betrachtet werden. Den Schwerpunkt bilden dabei die Bibliotheken zu den Bussystemen FlexRay und CAN sowie zum Gateway. Die Modellbibliotheken zu den Betriebssystemen sowie zum LIN-Bus sind als ergänzende Erweiterungen zu sehen.

4.5.1 Bibliotheken für die Kommunikationssysteme CAN und FlexRay sowie deren Kopplung

Für das Bussystem FlexRay existieren mehrere Modelle auf unterschiedlichen Abstraktionsebenen, daher ist die zugehörige Bibliothek in mehrere Abschnitte aufgeteilt. Zunächst finden sich Module, die vom Grad der Abstraktion unabhängig sind. Hierzu zählen das Modul CHI, welches die Schnittstelle zwischen dem Host und dem Communication Controller realisiert, sowie Basisfunktionen für das Applikationsmodell. Zu diesen Funktionen gehören das Senden und das Empfangen von Daten sowie die Initialisierung des Communication Controller. Die Verwendung der Basisfunktionen wird in einem einfachen Host-Modul demonstriert. Die zentrale Schnittstelle unterstützt einen einfachen Wechsel zwischen den verschiedenen Abstraktionsebenen

Die abstraktionsabhängigen Module gliedern sich in zwei Gruppen: Detailed und FrameBased. In der ersten Gruppe finden sich die detaillierten Umsetzungen für die Elemente Communication Controller (CC) und Channel. Das Modul CC beinhaltet eine spezifikationstreue Umsetzung der internen Mechanismen, Prozesse und der verteilten Zeitberechnung mit Hilfe von FSMs. Der Channel verbindet die einzelnen CCs und realisiert den Datenaustausch zwischen diesen auf der Ebene von Bit-Werten.

Die zweite Gruppe beinhaltet die abstrakten Varianten der Module CC und Channel. Durch den nachrichten-basierten Datenaustausch ist eine Vielzahl der protokollinternen Mechanismen obsolet. Diese werden nicht umgesetzt. Anstelle der verteilten Bestimmung der globalen Zeit, wird ein zentraler Zeitgeber verwendet. Für diesen zentralen Zeitgeber gibt es drei unterschiedliche Realisierungen. Durch einen einfachen Austausch dieses zentralen Zeitgebers kann zwischen den unterschiedlichen Abstraktionsebenen gewechselt werden.

Mit den vorhandenen Modulen lassen sich komplexe Kommunikationsarchitekturen realisieren. Zudem ist in der Bachelorarbeit von Gräbe (2007) ein passendes Tool zur Visualisierung der Kommunikation entstanden.

Für das Bussystem CAN existiert im Gegensatz zu FlexRay nur eine Abstraktionsebene. Die Bibliothek zum CAN-Bus enthält den Communication Controller und Channel für den nachrichten-basierten Datenaustausch. Die Logik zur Entscheidung über den Buszugriff enthält das Modul Channel. Die Schnittstelle zwischen CC und Host nennt sich beim CAN angelehnt an die Spezifikation durch das Modul LLC realisiert. Die Verwendung der vorhandenen Basisfunktionen für das Applikationsmodell, zum Senden und Empfangen von Daten, wird in einem einfachen Host-Modul demonstriert. Das CAN-Modell wurde zudem exemplarisch in das von Baumann (2009) entwickelte ESL-Target integriert.

Die Bibliothek zum Gateway nimmt eine Sonderstellung ein, da sie einzelne Module der beiden vorherigen Bibliotheken nutzt. Der Aufbau von heterogenen Systemen erfolgt unter Verwendung der CC und Channel der CAN und FlexRay Bibliotheken.

Es gibt eine Unterteilung in zwei logische Bereiche: GW-Core und GW-Host. Der erste Bereich enthält das Basismodul des Gateway und die zugehörigen Funktionsmodule zum Routing der Daten zwischen den einzelnen Netzen. Eine Erweiterung um weiter Gateway-Varianten ist durch einfache Substitution möglich.

Der zweite Bereich zerfällt in mehrere Teile und beinhaltet das Element Host für konkrete Bussysteme. Aktuell enthalten sind Module für CAN und FlexRay, wel-

che wiederum die Basisfunktionen aus den vorherigen Bibliotheken verwenden. Neben dem Senden und Empfangen von Daten haben die protokollspezifischen Host-Module die Aufgabe der Konvertierung der Bus-Nachrichten in das interne Gateway-Kommunikationsformat.

Die Gateway-Bibliothek mit ihren Modulen ermöglicht eine system- bzw. anwendungs-abhängige Erstellung eines Gateway-Modells, welches bei der Modellierung von heterogenen Systemen zur Kopplung eingesetzt werden kann.

4.5.2 Ergänzende Bibliotheken

Als Erweiterung zu diesen drei Basisbibliotheken können die Bibliotheken zu den Betriebssystem-Standards OSEK/OS und OSEKtime/OS sowie zum Kommunikationssystem LIN gesehen werden.

Die beiden Betriebssystemstandards sind in einer Bibliothek zusammengefasst. Für die einzelnen Betriebssystemfunktionen existieren Module (COM, FTCOM, Scheduler und Dispatcher). Diese werden als Beispiel in einem Modul zu einem Betriebssystem-Kern zusammengefasst. Die Verwendung und Anbindung von Umgebungsmodulen (Sensoren und Aktoren), Ressourcen und die Zusammenstellung einer Anwendung über Tasks werden an einem einfachen Beispiel demonstriert.

Die Bibliothek zum LIN-Bus enthält den Communication Controller und Channel für den nachrichten-basierten Datenaustausch. Den Communication Controller gibt es dabei in einer Master- und in einer Slave-Variante. Zusätzlich enthalten sind Basisfunktionen für die Umsetzung der Applikationsmodelle für Master und Slave. Für einen Master sind dies die Funktionen zur Konfiguration und Ablauf des Kommunikationsschedule und für den Slave das Senden und Empfangen von Nachrichten. Die Aufteilung nach Slave und Master bezieht sich dabei nach den LIN-Task und nicht nach den LIN-Knoten.

5 Transfer des Modellierungsansatzes in eine weitere Domäne: Mess- und Automatisierungstechnik

Die bisherige Demonstration des Modellierungskonzeptes beschränkte sich auf Anwendungsbereiche im Automotive-Segment. Für die dort eingesetzten Kommunikationsmechanismen und für die heterogene Struktur ist der Ansatz, wie zuvor auszugsweise dargestellt, geeignet. Im Folgenden soll die Verwendung in einer weiteren Anwendungsdomänen am Beispiel der Domäne Mess- und Automatisierungstechnik demonstriert werden. Dabei wird der Modellierungsansatz im Rahmen der Entwicklung, Analyse und Optimierung der Signal- und Datenverarbeitungseinheit einer Hochpräzisionsmessmaschine eingesetzt.

Ziel ist die Entwicklung einer Nanopositionier- und Nanomessmaschine (NPMM-200) mit einem Arbeitsbereich von $200 \times 200 \times 25\ mm^3$. Die grundsätzliche Funktionalität umfasst das Positionieren und Messen von Proben. Dabei wird eine Probe auf einer beweglichen Spiegelecke platziert. Die Spiegelecke lässt sich in drei Achsen bewegen. Über ein am metrologischen Rahmen befestigtes Tastsystem werden durch die Auslenkung des Tasters während der Bewegung Mess- und Steuerinformationen gewonnen.

Die Entwicklung kann in zwei getrennte Phasen unterteilt werden. In diesem zweistufigen Projekt geht es in der ersten Projektphase um eine komplette Neuentwicklung, das Projekt hat hier den Charakter eines Pionierprojektes (Kuster, 2011). Es handelt sich um ein interdisziplinäres Projekt, bei dem Wissen aus sehr unterschiedlichen Bereichen zusammenkommt. Das Projektlogo in Abbildung 5.0.1 visualisiert diese Interaktion zwischen verschiedensten Teilbereichen. Hinzu kommt, dass das Vorwissen sehr eingeschränkt ist.

Die auf die erste Phase aufbauende zweite Projektphase hat im Gegensatz dazu den Charakter eines Innovationsprojektes. Das bestehende Labormuster soll hier hinsichtlich der Leistungsfähigkeit und des Erweiterungspotentials untersucht und bewertet werden. Hier handelt es sich um eine Weiterentwicklung, ein gewisser Umfang an Vorwissen ist bereits vorhanden.

Gegenstand der folgenden Betrachtung ist die Entwicklung der Signal- und Datenverarbeitungseinheit für die Nanopositionier- und Nanomessmaschine, nachfolgend wird diese auf Grund des Bewegungsbereiches kurz als NPMM-200 bezeichnet. Mit der Nanopositionier- und Nanomessmaschine NPMM-200 wird in neue Bereiche bezüglich Messvolumen und Präzision vorgestoßen. Damit verbunden sind hohe Anforderungen an die Signal- und Datenverarbeitungseinheit.

Eine ausführliche Betrachtung von NPMMs sowie eine ausführliche Darstellung

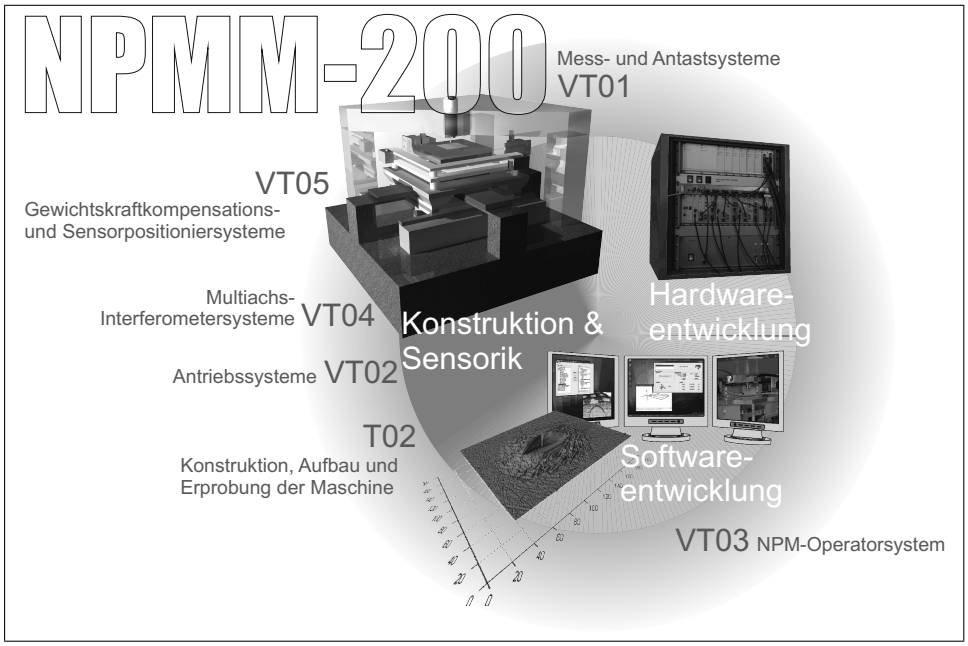


Abbildung 5.0.1: Projektlogo zur Darstellung des interdisziplinären Charakters

des SFB 622 erfolgt an dieser Stelle nicht. Es werden nur einige Auszüge zur Einordnung gegeben, diese finden sich im Anhang A.6. Weiterführende Informationen zu Nanopositionier- und Nanomessmaschinen sowie den grundlegenden Prinzipien der zu entwickelnden Maschine finden sich in (Hausotte, 2002) und (Hausotte, 2011).

5.1 Die Signal- und Datenverarbeitungseinheit der NPMM-200

Das zentrale Projektziel ist die Entwicklung einer Nanopositionier- und Nanomessmaschine (NPMM-200). In einem Transfer- und Verbundprojekt[1] zum Sonderforschungsbereich 622 „Nanopositionier- und Nanomessmaschinen“[2] (SFB 622) der Deutschen Forschungsgemeinschaft[3] (DFG) an der Technischen Universität Ilmenau sollte der Prototyp einer Maschine mit der folgenden Spezifikation[4] umgesetzt werden.

- Arbeitsbereich: $200 \times 200 \times 25\ mm^3$
- Auflösung: $0,08\ nm$
- Positionier-Reproduzierbarkeit: $1\ nm$
- 3D-Messunsicherheit: $< 30\ nm$
- Positioniergeschwindigkeit: $vmax = 30\ \frac{mm}{s}$

[1] http://www.tu-ilmenau.de/sfb622/transfer-und-verbundprojekt-npmm-200/ [08.01.2015]
[2] http://www.tu-ilmenau.de/sfb622 [08.01.2015]
[3] http://www.dfg.de [08.01.2015]
[4] http://www.tu-ilmenau.de/de/sfb622/transfer-und-verbundprojekt-npmm-200/zielstellung/ [08.01.2015]

Die Aufgabe umfasst die Entwicklung und Realisierung der Soft- und Hardwareplattform für die Informations- und Signalverarbeitung, die Regelung des Positioniertisches und der integrierten austauschbaren Tastsysteme unter Einbeziehung des Nullsensors (Sensor zur Referenzposition) sowie die Ablaufsteuerung. Ein großes Augenmerk liegt auf der sensornahen Messdatenverarbeitung.

Die gesamte Sensorik und Aktorik bestimmt die Anforderungen an die Signal- und Datenverarbeitungseinheit (SDPU - Signal- and Data Processing Unit). Hinzukommen die geforderten Taktraten sowie algorithmischen Anforderungen im Bereich Ablaufsteuerung, Trajektorien-Planung, Regelung sowie Vor- und Nachverarbeitung von Messwerten sowie von Signalen.

Bei dem experimentellen Aufbau handelt es sich um den Prototypen einer Nanopositionier- und Nanomessmaschine mit einem planarem Bewegungsbereich von $200 \times 200 \times 25\ mm^3$. Das System kann dabei für die Signal- und Datenverarbeitung in drei Bereiche aufgeteilt werden: Positioniersystem, Antastsystem und Regelungssystem. Im Anhang A.6 werden diese einzelnen Systeme etwas ausführlicher betrachtet.

Durch den hohen Grad an Parallelität innerhalb des Verarbeitungssystems bietet sich für die Realisierung eine verteilte Systemarchitektur an, welche eine parallele Verarbeitung ermöglicht. Somit ergibt sich grundsätzlich ein verteiltes eingebettetes System, wie es der vorgestellte Modellierungsansatz prinzipiell adressiert. Die zentrale Herausforderung bei der Echtzeitverarbeitung bildet die Erfüllung der zeitlichen Anforderungen durch die relativ hohen Taktraten. Direkt beeinflusst wird dies durch die Verteilung der Funktionen bzw. der Systempartitionierung, der Berechnung der Algorithmen sowie durch den Datenaustausch und die Kommunikation innerhalb des verteilten Systems.

Für die Realisierung der Signal- und Datenverarbeitungseinheit stehen standardisierte Hardware-Komponenten zur Verfügung. Diese sind PXI-Systeme bestehend aus Echtzeit-Controllern des Typs PXI-8108-RT (RT-Controller) und drei verschiedenen FPGA-Modultypen. Details zu den Hardwarekomponenten finden sich im Anhang A.8. Insbesondere durch die Möglichkeit der prozessnahen Verarbeitung innerhalb der FPGA-Module ergeben sich zahlreiche Varianten für die Realisierung des Systems.

Die Signal- und Datenverarbeitungseinheit der NPMM-200 baut auf standardisierten Hardware-Komponenten auf, stößt aber in Leistungsbereiche vor, für die diese einzelnen Komponenten nicht konzipiert sind.

5.2 Systementwicklung

Im Rahmen der Systementwicklung wird sich im Zusammenhang mit dem in den vorherigen Kapiteln vorgestellten Modellierungskonzept auf zwei Punkte beschränkt: die Systemarchitektur und die Kommunikation. Eine wesentliche Randbedingung ist, dass kein Vorwissen (keine Erfahrungen) über das System, die Hardware-Komponenten und die Leistungsfähigkeit existiert. Diese Einschränkungen haben erheblichen Einfluss auf die Modelle und den Gegenstand der modellbasierten Untersuchungen.

5.2.1 Entwicklungsprozess

Am Beispiel der Realisierung eines Regelungssystems wurde in (Zschäck et al., 2010) ein geeigneter domänenbezogener modellbasierter Systementwurf betrachtet, der insbesondere die Herausforderungen im Übergang zwischen plattform-unabhängigen zu plattform-spezifischen Modellen betrachtet. Eine ausführliche Vorstellung dieser Vorgehensweise findet sich in (Müller, Schwannecke & Fengler, 2012; Müller, 2013).

In der Regelungstechnik erfolgen das Regelungsdesign sowie die funktionale Validierung durch Simulation/Design Tools innerhalb eines Umgebungs- oder Anlagenmodells. Diese Tools mit ihren hardware-spezifischen Erweiterungen erlauben eine Darstellung der Regler-Implementierung auf der Modellierungsebene, während die Realisierung und der Hardware Deployment-Prozess automatisiert und versteckt durchgeführt wird (vgl. Abbildung 5.2.1).

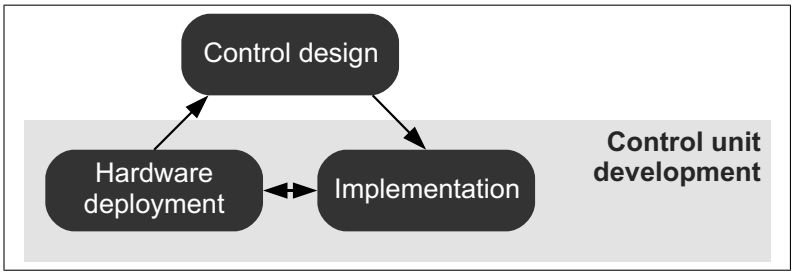


Abbildung 5.2.1: Control-Unit Entwicklungsprozess (vgl. Zschäck et al., 2010)

Dieser Entwicklungsprozess folgt der modellbasierten Entwicklungsmethode für eingebettete Systeme (Krasner, 2004). Unterschieden werden zwei unterschiedlich definierte Modelle - das plattform-unabhängige Modell (PIM) und das plattform-spezifische Modell (PSM) (vgl. Abbildung 5.2.2).

Das PIM ermöglicht die Erstellung einer strukturierten Verhaltensbeschreibung unabhängig von der Ausführung, Interaktionssemantik, oder der Hardware-spezifischen Implementierung (Henzinger & Sifakis, 2007). Im Gegensatz hierzu beschreibt das PSM die aktiven Objekte, die erforderlich sind, um das Verhalten des Reglers auf einer bestimmten Hardware zu implementieren und die Designtools bieten dabei Schnittstellen für die Low-Level-Hardware-orientierte Realisierung.

Im Rahmen der Entwicklung von verteilten Systemen sind die Implementierung des verteilten Hardware/Software-Systems sowie deren Validierung erforderlich. Die Verletzung von Leistungsanforderungen führt zur Optimierung der Implementierung, Architekturanpassungen oder dem Regelungs-Re-Design. Somit entsteht ein iterativer, ineffizienter und fehleranfälliger Prozess mit einer hohen Variabilität in den Bereichen Implementierung sowie Systemarchitektur und Regelungsdesign.

Der komplexe Übergang von PIM zu PSM beinhaltet das Finden einer optimalen Umsetzung. Bei verteilten Architekturen übersteigt dieses die Fähigkeit der Design-Tools. Ein reiner Prototyping-Ansatz ist sehr kostenintensiv, somit soll die Exploration des Entwurfsraums auf die Modellebene übertragen werden. Bei der Verfeinerung von Modellen müssen an einer bestimmten Stelle die hardware- und architektur-

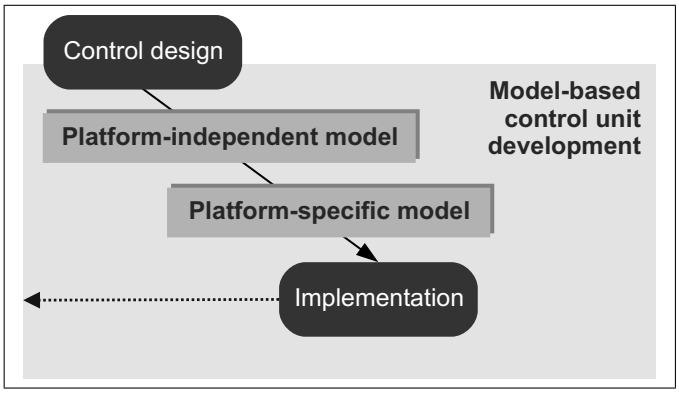


Abbildung 5.2.2: Modelle innerhalb des Entwicklungsprozesses (vgl. Zschäck et al., 2010)

spezifischen Eigenschaften mit in das Modell einfließen (Übergang PIM und PSM). Um eine optimale Hardware/Software-Zuordnung inklusive Auf- und Zuteilung von Funktionsmodulen zu Hardware-Elementen zur Erreichung eines optimalen Verhaltens des Gesamtsystems zu realisieren, wird der Model-Driven Development Prozess mit weiteren Aktivitäten ergänzt (vgl. Abbildung 5.2.3).

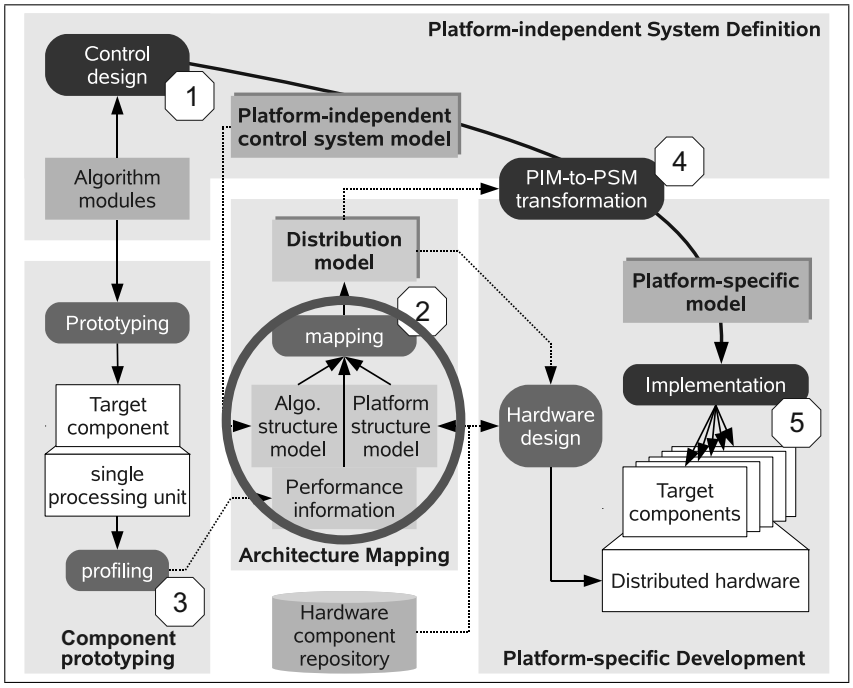


Abbildung 5.2.3: Schema des Entwufs- und Entwicklungsprozesses (Design Process) (vgl. Zschäck et al., 2010)

Der Regelungsentwurf (1) erfolgt in einer Modellierungsumgebung (bspw. MATLAB/Simulink), dieses PIM ist die Design-Eingabe für die folgenden Entwicklungsaktivitäten, welche zu einer plattform-spezifischen Realisierung (Unterstützung durch Toolchains von Hardwareherstellern) führen (5).

Vor der Erzeugung einer plattform-spezifischen Realisierung wird die PIM-to-PSM Transformation (4) durchgeführt. Diese Transformation basiert auf der zentralen Aktivität Architektur Mapping (2), wodurch ein Verteilungsmodell zur Anbindung des Implementierungsprozesses erzeugt wird.

Innerhalb des gekennzeichneten Mapping-Prozesses werden drei unterschiedliche Informationen verarbeitet: Informationen zur Algorithmen-Struktur aus dem PIM, Informationen zur Plattform-Struktur welche eine Strukturvariation erlaubt und Leistungsinformationen. Diese Leistungsparameter werden im Rahmen einer separaten Component Prototyping (3) Phase (Prototyping und Profiling) ermittelt. Die Optimierung der Gesamtarchitektur kann somit auf Modellebene erfolgen, ohne ein kostenintensives Prototyping des Gesamtsystems.

Ein Beispiel für das Prototyping und Profiling einzelner Komponenten wird in (Müller et al., 2013) anhand unterschiedlicher Realisierungsvarianten für einen Kalman Filter gegeben.

Neben diesem durchgängigen Entwurfsprozess gibt es unabhängig von einer konkreten Entwurfsaufgabe im Rahmen des modellbasierten Ansatzes weitere Möglichkeiten, welche mit dem Aspekt der Architekturbewertung in Verbindung stehen. Die Bewertung ist Teil des Mapping-Prozesses. Ein zusätzlicher Aspekt kann hier ergänzt werden: Performance Estimation. Hierunter wird die modell- und simulationsbasierte Gewinnung von Performance Parametern verstanden. Hierbei werden Strukturinformationen und Profilingdaten zur Ermittlung neuer und weiterer Performancedaten von einzelnen Komponenten verwendet.

5.2.2 Kommunikation

Für den Datenaustausch zwischen zwei PXI-Systemen gibt es zwei potentielle Kommunikationspfade: über Ethernet und über die digitale Schnittstelle der FPGA-Module. Der Datenaustausch innerhalb eines PXI-Systems erfolgt standardmäßig über den PCI-Bus. Hierfür können u.U. auch die digitalen Schnittstellen der FPGA-Module verwendet werden.

Für die externe Kommunikation (Kommunikation zwischen zwei PXI-Systemen) ist aufgrund der allgemein eingeschränkten Echtzeitfähigkeit der Ethernet-Kommunikation lediglich der Datenaustausch über die digitalen Schnittstellen der FPGA-Module möglich. Daher geht es im ersten Schritt um die Entwicklung einer geeigneten Schnittstelle, die sich für den Austausch der Messdaten zwischen PXI-Systemen eignet. Die Realisierung dieser Schnittstelle erfolgte im Rahmen der betreuten Diplomarbeit von Hausdörfer (2009). Für die Datenübertragung mit dem Ziel einer kurzen Latenz werden nachfolgende Anforderungen bzw. Randbedingungen aufgestellt bzw. ermittelt:

- Messdaten
  - verzögerungsarme Übertragung von 32 Messdaten
  - Das Auftreten der Messdaten kann unabhängig voneinander erfolgen
  - Die Zusammensetzung der Messdaten: 1 x 16 Bit, 25 x 32 Bit, 6 x 64 Bit
  - Gesamtgröße der Messdaten: 1200 Bit

  - maximale Frequenz des Auftretens eines Messwertes: 100 kHz
  - resultierende maximale Datenrate: 120.000.000 $bit/s = 120\ Mbit/s$

- Realisierung
  - Implementierung mit LabVIEW auf den FPGA-Modulen NI PXI-7813R / 7853R / 7854R
  - verfügbares Übertragungsmedium: maximal 40 digitale In-/Outputs (binäre Kanäle)

In der Diplomarbeit von Hausdörfer (2009) wurden ausgewählte Kommunikationsprotokolle mit unterschiedlichen Eigenschaften zur Vorbereitung einer kurzen Bewertung zur Einsetzbarkeit verglichen. Ausgewählt wurden die Feldbusse CAN und FlexRay sowie das Serial Peripheral Interface als einfacher synchroner serieller Bus (Motorola, Inc., 2004). Zudem wurden aus (Francis Rumsey, 1995) zwei Audioschnittstellen ausgewählt, ein serielles 2-Kanal Interface mit einem synchronen Zeitmultiplexing sowie ein paralleles Mehrkanalinterface (das von Mitsubishi entwickelte Pro Digi Format) welches als Zugriffsverfahren ein Space Division Multiplex (SDM) einsetzt. Ergänzt werden diese um den Token Ring (Conrads, 1996) und Ethernet (Stallings, 2004) aus dem Anwendungsbereich der lokalen Netzwerke (LAN) sowie Synchronous Digital Hierarchy (SDH) (Wilde, 1999) und Asynchronous Transfer Mode (ATM) (Riggert, 1998) aus dem Anwendungsbereich der Weiterverkehrsnetzwerke (WAN).

Die Anwendung erfordert eine einfache Punkt-zu-Punkt Verbindung. Daher genügt die Betrachtung des Physical und des Data Link Layer. Relevante Kriterien sind:

Paketgröße Eine minimale Verzögerung erfordert eine konstante Paketgröße. Variabilität führt an dieser Stelle zu unnötigem Overhead.

Multiplexing / Multiple Access Eine optimale Nutzung des Übertragungsmediums (Kombination Raum- (SDM) und Zeitmultiplexing (TDM))

Taktübertragung Die Verfügbarkeit von 40 digitalen Kanälen erlaubt die Verwendung einer separaten Taktleitung. Die Verwendung eines selbsttaktenden Leitungscodes erhöht den Aufwand und die Komplexität.

Übertragungsmedium Als Übertragungsmedium stehen 40 digitale Kanäle zur Verfügung. Protokolle mit einem Shared-Medium erfüllen bei Beschränkung der Netzgröße auf zwei Teilnehmer entsprechende Eigenschaften.

Eine kurze Gegenüberstellung der genannten Protokolle mit Berücksichtigung der vier Kriterien ist in Tabelle 5.1 dargestellt.

Die sehr spezifischen Anforderungen der Anwendungen werden von keinem der genannten Protokolle in ausreichendem Umfang erfüllt. Ein Defizit der Protokolle (mit Ausnahme des SPI-Protokolls) ist der nicht näher betrachtete Overhead der Protokolle, für den geforderten sehr einfachen Austausch von Messdaten mit einer geringen Latenz. Da das Anwendungsfeld der Protokolle mehr als die Übertragung von einfachen Messdaten erfordert, ist der gesamte Prokollaufbau komplexer. Einzelne Basismechanismen der Protokolle können in einem selbstentwickelten Protokoll allerdings verwendet werden.

Tabelle 5.1: Gegenüberstellung ausgewählter Kommunikationsprotokolle (vgl. Hausdörfer, 2009)

	Paket-größe	Multiplexing/ Multiple Access/	separate Taktleitung	Übertragungs-medium
Serial Peripheral Interface	konstant (+)	kein	ja (+)	exklusiv: 1 Takt-, 2 Datenleitungen (o)
Serielles 2-Kanal Interface	konstant (+)	synchrones TDM	nein (-)	exklusiv (o)
Mitsubishi Interface	konstant (+)	SDM	ja (+)	exklusiv 1 Taktleitung 32 Datenleitungen 2 Steuerleitungen (o)
CAN	variabel (-)	CSMA/CA	nein (-)	geteilt (-)
FlexRay	konstant (+)	TDMA	nein (-)	geteilt (-)
Token Ring	variabel (-)	Token Passing	nein (-)	exklusiv zwischen jeweils 2 Knoten im Ring (o)
Ethernet	variabel (-)	CSMA/CD	nein (-)	geteilt (-)
SDH	konstant (+)	synchrones TDM	zentraler Takt (o)	exklusiv (o)
ATM	konstant (+)	asynchrones TDM	nein (-)	

5.2.2.1 Kommunikationskonzept für Multi-FPGA-Systeme

In (Müller, 2013; Krahn, 2010; Müller, Brandel, Krahn & Fengler, 2011) wird ein Interface namens SHIVA (Shared Memory Interface for versatile Application) für die Intermodulkommunikation für ein Multi-FPGA-System vorgestellt. Die Systemstruktur der Signal- und Steuerungseinheit der NPMM-200 kann teilweise als Multi-FPGA-System gesehen werden und ist somit grundsätzlich für die Verwendung von SHIVA geeignet. Die Eignung und Verwendung von SHIVA für die Kommuniktion wird nachfolgend geprüft.

5.2.2.1.1 Charakterisierung des Kommunikationskonzeptes SHIVA Die Kommunikation und der Datenaustausch können nie losgelöst von ihrem Umfeld betrachtet werden. Einschränkungen durch die Hardware und Anforderungen durch die Anwendung haben immer einen Einfluss auf die Charakteristik. Im Falle von SHIVA gilt hinsichtlich der Anwendungsebene, dass die Applikation in Hardware realisiert wird

und dabei aufgrund des Ressourcenbedarfs mehrere FPGA-Module eingesetzt werden müssen auf welche einzelne Funktionen der Applikation verteilt werden.

Die Entwicklung des Kommunikationskonzeptes SHIVA berücksichtigt die spezifischen Eigenschaften der Zielhardware. Bei der zugrunde liegenden Hardware (Multi-FPGA-System basierend auf GECKO3-Modulen[5]) kann für den Datenaustausch einer der beiden vorhandenen Busverbinder genutzt werden. Dieser bietet im konkreten Fall 116 Signalleitungen zur freien Verfügung.

Als Basis für den Datenaustausch wird ein „Virtual Shared Memory“ Konzept genutzt. Hierbei wird der globale Speicher auf den Block-RAM der einzelnen FPGA-Module verteilt. Der Datenzugriff erfolgt somit über den lokalen Teil des globalen Speichers oder mittels Kommunikation über den Bus durch Zugriff auf den Speicher eines anderen Moduls.

5.2.2.1.2 Kommunikationsbus Der parallele bidirektionale Bus setzt sich aus den Komponenten Daten-, Adress- und Steuerbus sowie der Zugriffssteuerung über den Tokenbus zusammen, die Datenbreiten (Anzahl der Signalleitungen) der Busse finden sich in Tabelle 5.2.

Tabelle 5.2: Aufbau des Busses zur Datenkommunikation

Bezeichnung	Anzahl der Leitungen	
Datenbus	64	Ein Datum besteht aus 64 Bit
Adressbus	$n = \lceil \mathrm{ld}(\text{Anzahl der Module} + 1) \rceil + \lceil \mathrm{ld}(\text{Anzahl der RAM-Blöcke}) \rceil + 8$	Die Adresse setzt sich zusammen aus der Knoten ID, der Blockadresse und der Datenadresse
Steuerbus	2	
Tokenbus	$k = \lceil \mathrm{ld}(\text{Anzahl der Module} + 1) \rceil$	Das Token repräsentiert das Senderecht eines Moduls und ist mit dessen Identifier gleichzusetzen

Die Leistungsfähigkeit der Kommunikation wird durch den auf einem Token Passing Verfahren basierenden Zugriffsmechanismus bestimmt. Die Vergabe des Senderechtes erfolgt in einem sehr einfachen Verfahren. Das Senderecht wird in einem Zyklus jedem Modul unabhängig vom Sendewunsch für eine konstante Zeit durch einen Arbiter zur Verfügung gestellt. Es handelt sich demnach um ein Single-Master-System mit einem festen Kommunikationsschedule. Unter Annahme einer maximalen Buslast wird bei dem erzielten Bustakt von 50 MHz eine Datenübertragungsrate von $1,6$ $Gbit/s$ für das Gesamtsystem erreicht. Diese Datenrate teilen sich alle beteiligten Module.

5.2.2.1.3 Bewertung des Konzeptes zur Nutzung in der NPMM-200 Eine zentrale Schwäche dieser Buskommunikation ist die statische Vergabe des Buszugriffes

[5] `http://labs.ti.bfh.ch/gecko/` [08.01.2015]

nach einem Round-Robin-Mechanismus. Hiervon profitieren Funktionspartitionierungen einer Anwendung mit einem sehr ausgeglichenen Kommunikationsbedarf. Im Allgemeinen ist dies nicht gegeben. Durch eine Anpassung des Mechanismus zur Verteilung des Senderechtes lässt sich dieser Sachverhalt mit einem gewissen Entwicklungsaufwand den Anforderungen an das Multi-FPGA-System der NPMM-200 anpassen.

Eine Vielzahl von Eigenschaften widerspricht der Verwendung von SHIVA innerhalb der Signal- und Datenverarbeitungseinheit der NPMM-200. Das Anwendungsszenario für SHIVA sieht vor, dass die Anwendung bzw. insbesondere auch einzelne Teilfunktionen auf mehrere FPGA-Module verteilt werden und zur Verarbeitung zwischen den einzelnen Teilfunktionen der Austausch von einzelnen Werten notwendig ist. Dagegen zerfällt die Steuerung der NPMM-200 in einzelne Komponenten, die einen feingranularen Datenaustausch nicht benötigen. Der Datenaustausch zwischen den einzelnen Komponenten erfolgt in Datenblöcken, welche aus mehreren Messwerten bestehen. Der Datenfluss im gesamten System erfolgt im Allgemeinen unidirektional. Ein wechselseitiger Datenaustausch ist nicht erforderlich. Enge Abhängigkeiten zwischen den Komponenten sollen durch die Partitionierung vermieden werden, und es gibt keine Verteilung von Teilfunktionen einer Komponente auf mehrere Module. Diese Sachverhalte sprechen mehr für eine Punkt-zu-Punkt-Verbindung zwischen einzelnen Komponenten.

Neben diesen Randbedingungen, welche der konkreten Anwendung zuzuordnen sind, ergeben sich aus den verfügbaren oder ausgewählten Hardwarekomponenten weitere Eigenschaften, die gegen eine Verwendung von SHIVA sprechen. Während die Zielhardware von SHIVA den direkten Zugriff auf FPGA-Schnittstellen erlaubt, ist der Datenaustausch zwischen Modulen im Prototypen der NPMM-200 lediglich über „externe“ digitale Schnittstellen mit max. 40 MHz möglich. Dies hat eine reduzierte Leistungsfähigkeit von SHIVA zur Folge. Die maximale Übertragungsrate sinkt folglich um 20%.

Ein gravierenderes Problem stellen die Anforderungen an die Anzahl der Signalleitungen dar, SHIVA benötigt in einer einfachen Konfiguration mit zwei Knoten und einem Block-RAM insgesamt 80 digitale Signale. Davon werden 64 für die Daten und 16 für die Steuerung der Kommunikation genutzt. Die Schnittstellen der FPGA-Module der NPMM-200 besitzen pro Verbinder lediglich 40 digitale Schnittstellen, somit könnten maximal 24 Bit Daten übertragen werden. Die maximale Übertragungsrate sinkt somit nochmals um $62,5\%$. Statt der $1,6$ $GBit/s$ liegt sie nur noch bei 480 $MBit/s$.

Die resultierende Leistungsfähigkeit ist somit deutlich schlechter als die Leistungsfähigkeit der speziell anhand der Anforderungen der NPMM-200 entwickelten Kommunikationsschnittstellen, die in dem folgenden Abschnitt vorgestellt werden.

5.2.2.2 Entwicklung eigener Protokolle

Die Entwicklung von spezifischen anwendungsfallorientierten Protokollen war wie zuvor dargestellt notwendig. Die entwickelten Protokolle werden nachfolgend vorgestellt. Betrachtet werden bei diesen sehr einfachen Protokollen jeweils nur Physical Layer und Data Link Layer. In der Entwicklungsphase erfolgt eine Verifikation einzelner Mechanismen mit geeigneten Modellen. Der Entwicklungszeitraum der dargestellten

Mechanismen umfasst beide Projektphasen.

5.2.2.2.1 Physical Layer Innerhalb des Physical Layer werden drei unterschiedliche Varianten betrachtet, die erste Variante mit einer maximalen Datenrate (PLA - Physical Layer Variante A), eine zweite Variante (PLB - Physical Layer Variante B) mit einer standardisierten Datenbreite und eine dritte Variante (PLC - Physical Layer Variante C), welche zur Bestimmung der Datenwerte ein Majority Voting durchführt. Bei allen drei Varianten erfolgt die Datenübertragung asynchron und es existiert zu den Datenleitungen eine separate Taktleitung.

Variante A mit maximaler Datenrate (PLA - 1 Taktleitung, 39 Datenleitungen) Als Basis für den Physical Layer dient das parallele Mitsubishi Interface, es wird allerdings eine Taktleitung zusammen mit 39 Datenleitungen verwendet. Eine Empfangsinstanz tastet das Signal auf der Taktleitung ab und speichert bei Erkennung einer Flanke die 39 binären Werte der Datenleitungen ab. Eine Sendeinstanz legt gleichzeitig die Datenwerte auf die Datenleitungen und wechselt den Pegel der Taktleitung. Diese Protokoll-Variante ist explizit für die Übertragung eines spezifischen Messdatenblocks ausgelegt, welcher auf 16 x 39-Bit aufgeteilt wird.

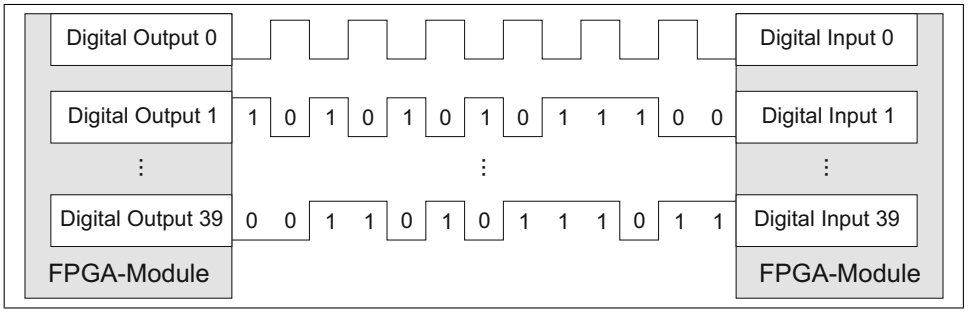


Abbildung 5.2.4: Physikalische Verbindung zwischen Sender und Empfänger (PLA)

In Abbildung 5.2.4 ist die Zuordnung der Takt- und Datenleitungen dargestellt. Die Zustandsdiagramme für eine asynchrone Sende- und eine Empfangsinstanz für eine 16×39-Bit Übertragung zeigt Abbildung 5.2.5.

Der Aufbau und die Funktionalität dieses Physical Layer sind spezifisch auf den zu übertragenden Messdatenblock zugeschnitten, zur Übertragung anderer Datenblöcke müsste dieser Layer angepasst werden. Diese Einschränkung wird durch die folgenden beiden Varianten behoben.

Variante B mit standardisierter Datenbreite (PLB - 1 Taktleitung, 32 Datenleitungen) Für die 32-Bit Kommunikationsvariante wird die digitale I/O-Leitung 0 (DIO 0) als Taktsignal und die digitalen I/O-Leitungen 1 bis 32 (DIO1 - DIO32) zur Übertragung der Datenbits verwendet (vgl. Abbildung 5.2.6). Die verbleibenden sieben digitalen Kanäle eines Connectors des FPGA-Moduls stehen für zusätzliche Funktionen zur Verfügung.

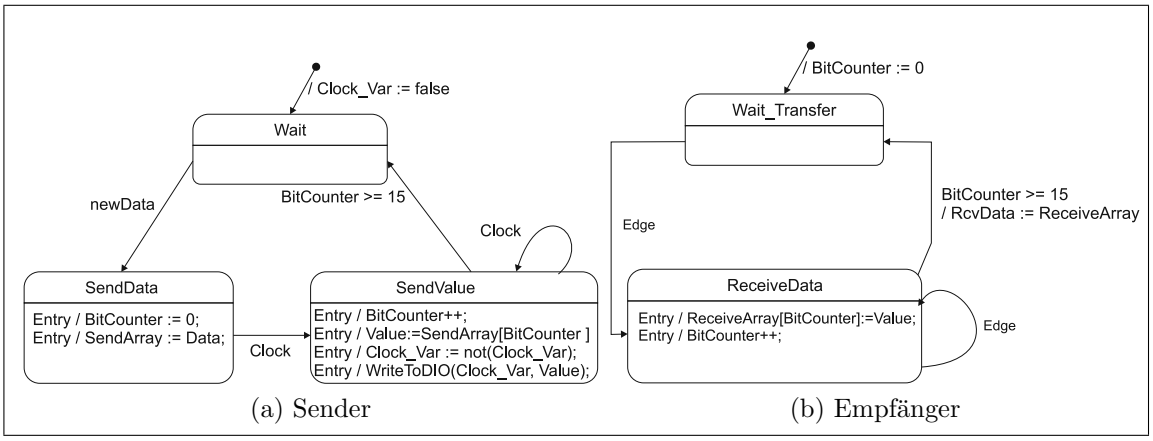


Abbildung 5.2.5: Zustandsdiagramme für PLA Sende- und eine Empfangsinstanz

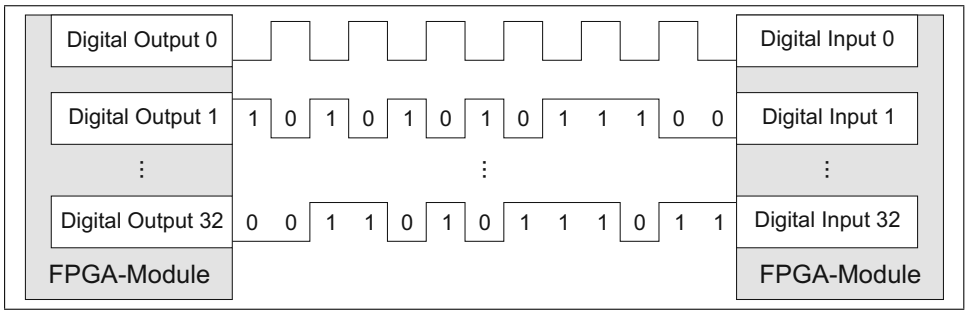
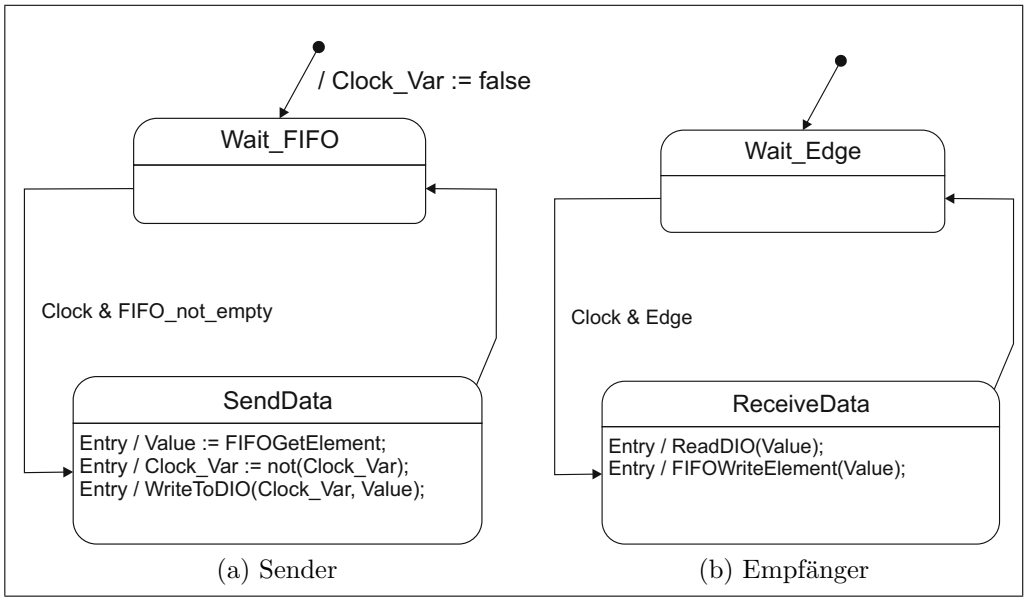


Abbildung 5.2.6: Physikalische Verbindung zwischen Sender und Empfänger (PLB)

Durch die feste Verwendung von 32 Bit (uInt32) ist eine allgemeinere flexible Verwendung der Übertragungsschnittstelle möglich. Der Zugangspunkt (Schnittstelle) zur übergeordneten Schicht wird über eine FIFO (Datenspeicher innerhalb des FPGA-Moduls) realisiert. Die Sendeinstanz nimmt einen 32-Bit-Wert aus der FIFO, legt diesen Wert auf die Datenleitungen und ändert den Pegel auf der Taktleitung. Die Empfangsinstanz arbeitet parallel und wartet auf einen Pegelwechsel der Taktleitung, liest nach erfolgtem Pegelwechsel die Datenleitungen ein und schreibt den 32-Bit-Wert anschließend in eine Empfangs-FIFO. In der Darstellung 5.2.7 sind die jeweiligen Zustandsdiagramme für Sende- und Empfangsinstanz zu finden. Der Sender arbeitet dabei ebenso wie bei PLA standardmäßig mit einem Takt von 16 MHz und der Empfänger mit einem Takt von 40 MHz (Taktung des Automaten).

Variante C mit Majority Voting (PLC - 1 Taktleitung, 32 Datenleitungen) Die dritte Variante des Physical Layer arbeitet mit einem variablen Sendetakt von bis zu 40 MHz in Verbindung mit einem Majority Voting Mechanismus. Die maximale Übertragungsrate steigert sich damit im Vergleich zu PLB von 512 $Mbit/s$ auf 1280 $Mbit/s$.

Die Sendeinstanzen von PLB und PLC sind identisch und unterscheiden sich lediglich durch die variable Taktrate. Die Empfangsinstanz beinhaltet einen Mechanismus zur Mehrfachabtastung des Signals und arbeitet mit einem deutlich höheren Takt im



(a) Sender (b) Empfänger

Abbildung 5.2.7: Zustandsdiagramme für PLB Sende- und eine Empfangsinstanz

Vergleich zur Sendeinstanz. Basierend auf einem Majority-Voting wird aus mehreren Abfragewerten eines Übertragungsvorgangs ein Datenwert bestimmt. Zunächst wird in jedem Empfangstakt die Taktleitung eingelesen, wird hier eine Flanke registriert werden in den folgenden drei Empfangstakten die Datenleitungen eingelesen. Im dritten Empfangstakt nach der Flankenerkennung des Taktsignals wird mittels bitweisen Majority-Votings der empfangene Datenwert bestimmt und in die FIFO geschrieben. In Abbildung 5.2.8 wird das Timing dieses Mechanismus an einem Beispiel verdeutlicht.

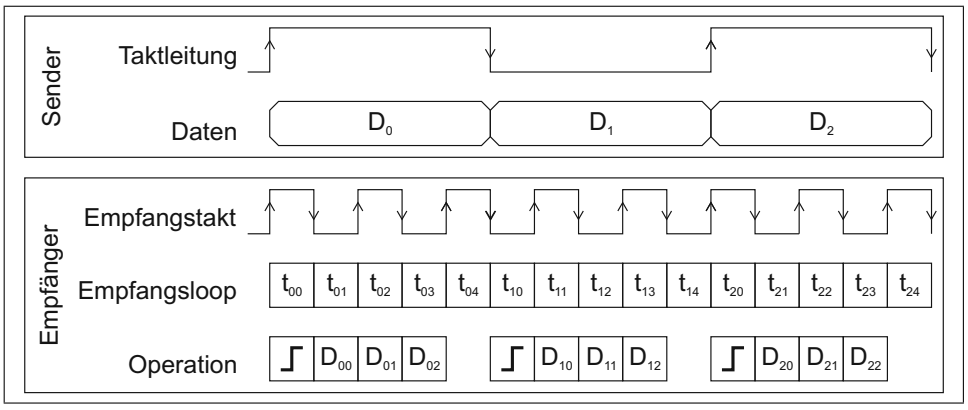


Abbildung 5.2.8: Timing des Datenempfangs PLC

Die beiden letzten Varianten lassen sich leicht skalieren und mit weiteren Datenbreiten realisieren zum Beispiel 16 Bit und 8 Bit.

5.2.2.2.2 Data Link Layer

Data Link Layer Variante A (DLLA für PLA) Die DLL-Variante A arbeitet auf dem Grundprinzip der seriellen, digitalen Audioschnittstellen. Für das Anwendungsszenario ist die Voraussetzung, dass alle Daten zu Beginn des Sendevorganges bereitstehen. Verwendet wird eine Mischung aus synchronem Zeitmultiplexing (TDM) und Raummultiplexing (SDM). Die Identifikation der Frames erfolgt über das zeitliche Auftreten des physikalischen Kanals, somit kann eine explizite Adressierung entfallen.

Ein Frame besteht aus 16 Bit und wird jeweils auf einer Datenleitung versendet. Die Messwerte werden in Frames segmentiert. Somit besteht ein Sendezyklus zur Übertragung eines Messwertblocks aus zwei Durchgängen bei denen jeweils 39 Frames übertragen werden.

Data Link Layer Variante B (DLLB für PLA) Das prinzipielle SDH-Multiplexing dient als Basis für die Variante B. Jedem Frame wird ein Valid-Flag zur Kennzeichnung eines gültigen Wertes zugeordnet, somit müssen zu Beginn eines Sendevorgangs nicht alle Messdaten zur Verfügung stehen.

Der Unterschied zu Variante A besteht darin, dass ein Frame aus 17 Bit besteht. Somit besteht ein Sendezyklus aus 2 x 39 x 17 Bit. Die Verwendung der zusätzlichen Valid-Flag erhöht zwar den Overhead steigert aber die Flexibilität.

Data Link Layer Variante C (DLLC für PLA) Bei Variante C wird das Prinzip von virtuellen Verbindungen verwendet, ähnlich wie bei der ATM-Zellenvermittlung. Eine Zelle hat eine Größe von 16 Bit. Bei der vorliegenden Punkt-zu-Punkt Verbindung ist kein Verbindungsaufbau notwendig.

Die Komplexität und Flexibilität ist durch ein asynchrones Multiplexing deutlich höher. Der Zusammenhang Eingangsreihenfolge und Messwertzugehörigkeit ist nicht mehr gegeben.

Ein Sendezyklus besteht nun aus nur einem Durchlauf (39 x 17 Bit). Den Frames wird ein Valid Flag eines Messwertes vorangestellt (bzw. ein Dummy-Bit). Diese Flags haben eine feste Anordnung und dienen zur Adressierung.

Die Übertragung ist in zwei Bereiche aufgeteilt: Primär- und Sekundärwerte. Bei der Übertragung werden nicht gültige Primär- durch Sekundärwerte ersetzt. Eine Übertragung von Sekundärwerten endet spätestens nach zwei Sendezyklen.

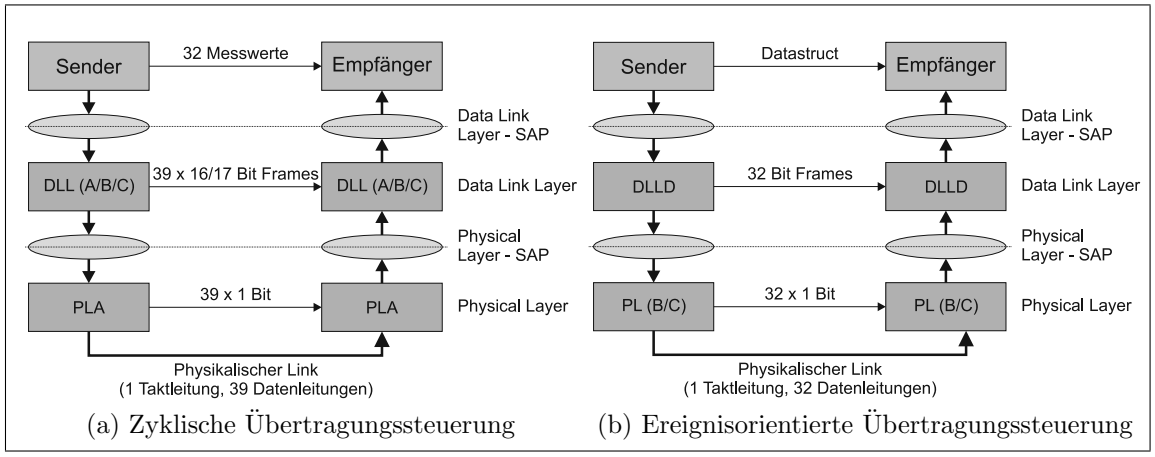
Data Link Layer Variante D (DLLD für PLB und PLC) Die vierte Variante des DLL eignet sich für die Nutzung in Verbindung mit den Physical Layern PLB und PLC. Der einfache, flexible Einsatz steht hier im Vordergrund. Aufgabe ist die Verbindung von Applikationsebene und Physical Layer, dazu wird die zu übertragende Datenstruktur in 32-Bit große Frames aufgeteilt. Die einzelnen Frames werden PLB oder PLC übergeben, welche wiederum ein einzelnes Frame auf die 32 Datenleitungen legen.

Die Segmentierung der Datenstrukturen sowie die zeitliche Abfolge der Daten wird a priori festgelegt und ist Sender und Empfänger bekannt, somit müssen keine zusätzlichen Informationen übertragen werden.

Durch die Verbindung von PLB oder PLC und DLLD entsteht eine flexible, leistungsfähige, konfigurierbare Kommunikationsschnittstelle zum Austausch von Daten zwischen FPGA-Modulen innerhalb der Systemarchitektur der Signal- und Datenverarbeitungseinheit der NPMM-200.

5.2.2.2.3 ISO-OSI-Schichtenmodell der Basisvarianten Grundsätzlich stehen innerhalb der Signal- und Datenverarbeitungseinheit für die Kommunikation zwei Basisvarianten zur Verfügung, eine Bitraten-maximale Variante zur zyklischen Übertragung von Messwertblöcken und eine flexiblere Variante zur ereignisgesteuerten Übertragung von Datenstrukturen.

In Abbildung 5.2.9 werden die beiden Varianten in einer ISO-OSI-Schichtenmodell orientierten Darstellung gegenübergestellt. Im Weiteren wird ausschließlich die zweite Variante zur ereignisgesteuerten Übertragung von Datenstrukturen betrachtet.



(a) Zyklische Übertragungssteuerung (b) Ereignisorientierte Übertragungssteuerung

Abbildung 5.2.9: Gegenüberstellung der Basisvarianten

5.2.2.3 Protokoll Erweiterung - Fehlererkennung und Fehlerbehebung

Beide vorgestellten Basisvarianten beinhalten keine Sicherheitsmechanismen. Abhängig von spezifischen Anforderungen und damit verbundenen Variationen im Bereich der Signal- und Datenverarbeitung werden für die zweite Basisvariante nachfolgend Ergänzungen zur Erhöhung der Übertragungssicherheit (Mechanismen zur Fehlererkennung und -behebung, kurz Kodierung) vorgestellt. An dieser Stelle wird auf eine Betrachtung der Grundlagen zu Sicherheitsmechanismen und Kodierung bei der Kommunikation verzichtet und auf (Furrer, 1981; Sweeney & Höfler, 1992; Gravano, 2001; Swoboda, 1973) verwiesen.

Wie in Abbildung 5.2.10 dargestellt, lassen sich diese je nach Anforderung der Gesamtapplikation direkt in den Ablauf der Kommunikation integrieren. Die Komplexität (Funktionsumfang) und somit der Ressourcenbedarf (Platzbedarf auf dem FPGA-Modul und Zeitbedarf der Kommunikation) erhöhen sich. Reine Fehlererkennungsmechanismen bringen keine Vorteile, da ein erkannter Fehler zum Herunterfah-

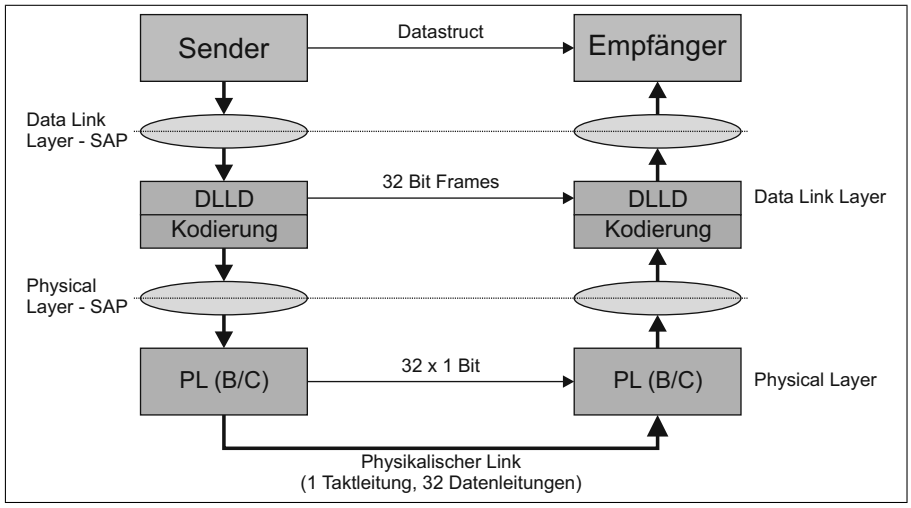


Abbildung 5.2.10: Schichtenmodell der Kommunikation mit integrierter Kodierung

ren des Systems führt, daher werden ausschließlich Mechanismen zur Fehlerbehebung betrachtet. Als Beispiel für eine Kodierung werden hier Hamming-Codes in zwei Varianten herangezogen, zum einen als Kodierung über die Datenleitungen (vertikale Kodierung) und zum anderen als Kodierung auf einer Datenleitung (horizonale Kodierung).

5.2.2.3.1 Vertikale Kodierung Bei der ersten Variante erfolgt die Kodierung wie in Abbildung 5.2.11 dargestellt über ein paralleles Datenbündel (vertikale Datenkodierung). Ein Vorteil der vertikalen Kodierung ist, dass diese für alle Schnittstellen identisch genutzt werden kann, da sie unabhängig von der versendeten Paketgröße ist. Die Realisierung dieser Variante der Kodierung erfolgte in der betreuten Diplomarbeit von Timmler (2011).

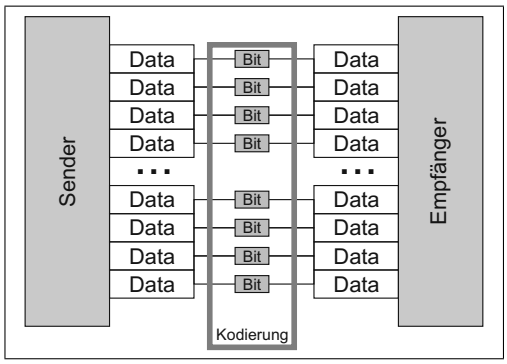


Abbildung 5.2.11: Vertikale Kodierung der Daten

Die Übertragung der Daten erfolgt über die 32 parallelen digitalen Kanäle. Daher eignen sich ein (31,26)- und ein (63,57)-Hamming-Code mit einer 1-Bit-Fehler-Korrekturfähigkeit für die Codierung. Die jeweiligen Generatorpolynome (x^5+x^2+1, x^6+x+1) werden für die Erzeugung der kodierten Nachrichten und der Kontrollstellen

genutzt. (Swoboda, 1973)

Die zu übertragenden DLL-Frames (32-Bit) müssen zunächst segmentiert werden, um anschließend kodiert zu werden. Nach der Übertragung erfolgt die Decodierung. Hier wird ein Syndrom berechnet, der Fehlervektor bestimmt und ggf. die Korrektur vorgenommen.

Der Vorteil dieser Variante ist, dass durch die Kodierung ein recht geringer Overhead entsteht. Direkt damit verbunden ist allerdings die aufwändige Segmentierung der Daten.

5.2.2.3.2 Horizontale Kodierung Bei der vertikalen Kodierung ist die Segmentierung sehr aufwändig und schlägt sich negativ auf die Übertragungsleistung nieder. Dieser Nachteil wird durch eine horizontale Kodierung wie in Abbildung 5.2.12 dargestellt behoben. Hierbei werden nicht die einzelnen Frames kodiert, sondern es werden die Bits einer Datenstruktur, welche einer Datenleitung zugeordnet sind zusammengefasst und diese kodiert.

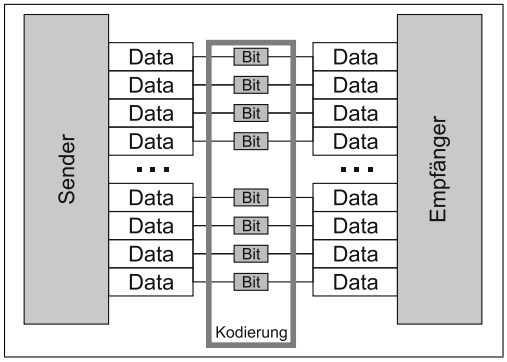


Abbildung 5.2.12: Horizontale Kodierung der Daten

Die einzelnen Schritte werden anhand eines kurzen Beispiels (Abbildung 5.2.13) mit einer Datenstruktur welche drei u64-Werte und fünf u32-Werten beinhaltet verdeutlicht. Genutzt wird in diesem Beispiel ein (15,11)-Hamming-Code.

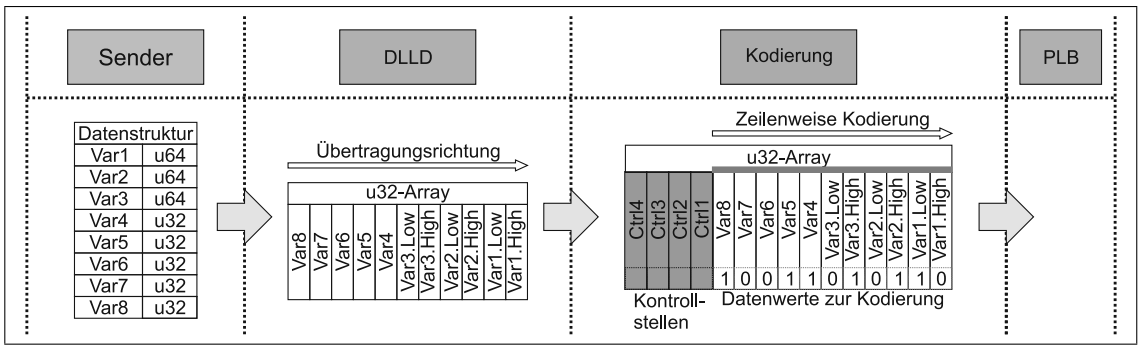


Abbildung 5.2.13: Beispiel für die horizontale Kodierung der Daten

Zunächst wird die Datenstruktur in u32-Werte segmentiert, diese werden in einem Array gespeichert. Dieses Array wird anschließend kodiert. Bei der Kodierung wird

das u32-Array als zweidimensionales binäres Array interpretiert und neu aufgeteilt. Die Zeilen dieses Arrays enthalten nun 11 Datenbits, diese werden entsprechend des (15,11)-Hamming-Codes zeilenweise kodiert und die Kontrollstellen in die Bits 12-15 geschrieben. Das neu entstandene Array besteht nun aus 15 Elementen.

Der Vorteil dieser Variante ist die einfache Segmentierung sowie die elementweise Kodierung entlang der Übertragungsrichtung. Im Vergleich zu dem (31,26)-Hamming-Code ist in diesem Beispiel lediglich eine zusätzliche Übertragung notwendig. Diese Verzögerung wird dadurch kompensiert, dass die Kodierung elementweise parallel zum Sendevorgang durchgeführt werden kann, während bei der vertikalen Kodierung vor der Übertragung die 26-Bit zunächst verarbeitet werden müssen.

5.2.2.3.3 Realisierung einer Komponente für die horizontale Kodierung für ein FPGA-Modul mit LabVIEW In Abbildung 5.2.13 ist der schematische Ablauf der horizontalen Kodierung der Daten dargestellt. Dabei übermittelt der Sender eine Datenstruktur bestehend aus acht Elementen welche zunächst in ein Array vom Datentyp uInt32 umgewandelt werden. Anschließend erfolgt die Kodierung der Daten und die Übertragung zur Komponente, welche die Übertragung der Daten realisiert. Die Realisierung der zugehörigen Standard-Komponenten für die Kommunikation findet sich in Abschnitt 5.2.2.4. Nachfolgend wird die Realisierung einer LabVIEW-Komponente (Sub-VI), welche den Schritt der Kodierung umfasst, kurz dargestellt. Als Beispiel wird an dieser Stelle ein VI (Abbildung 5.2.14) ausgewählt, welches die horizontale Kodierung mit einem (15,11)-Hamming-Code realisiert.

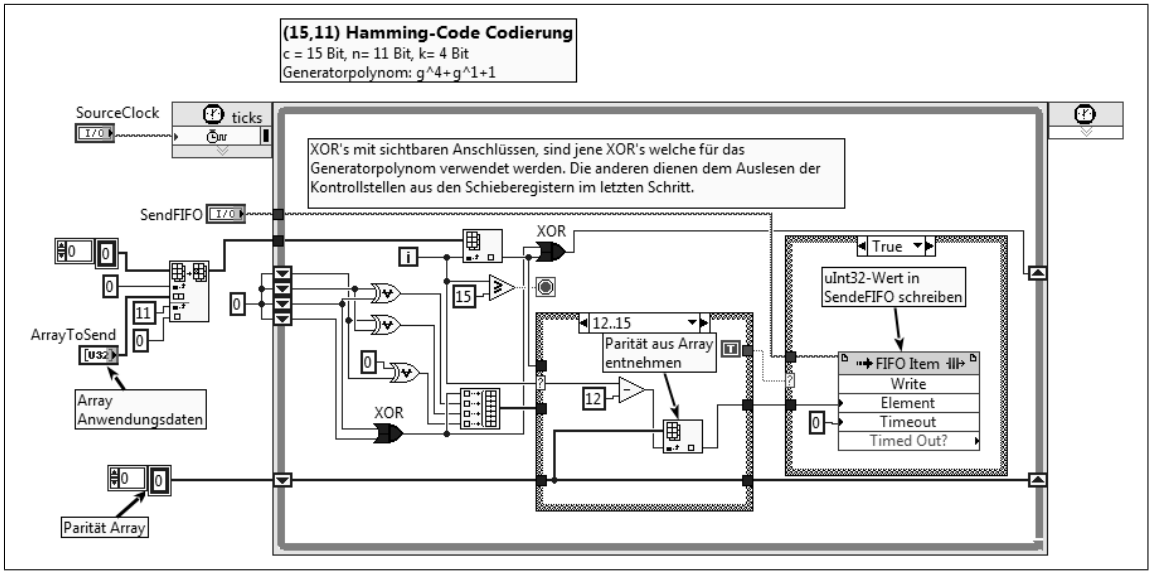


Abbildung 5.2.14: FPGA-Realisierung der horizontalen Kodierung mit einem Hamming-Code (15,11) mit LabVIEW

Das Kodierungsmodul wird als Sub-VI in ein übergeordnetes VI eingebunden. Die Nutzerdaten (11 Werte vom Typ uInt32) werden übergeben und in die FIFO zur Übergabe der Daten zwischen dem Data Link Layer und dem Physical Layer einge-

tragen, parallel werden dazu die Kontrollstellen bestimmt und nach der Generierung ebenfalls in die FIFO eingetragen.

Das VI besitzt drei Eingänge: ein Array der Größe 11 (*ArrayToSend*) über welches die zu sendenden Daten übergeben werden, eine FIFO-Referenz (*SendFIFO*) über welche die FIFO zur Wertübergabe für die physikalische Übertragung festgelegt wird und eine Clock-Referenz (*SourceClock*) über welche der Ausführungstakt der Schleife festgelegt wird.

Die Schleife wird insgesamt 16-mal ausgeführt, während der ersten elf Durchläufe werden die Nutzdaten in die FIFO eingefügt und die Kontrollstellen berechnet. Im zwölften Durchlauf werden die erzeugten Kontrollstellen aus den Registern entnommen und in den folgenden vier Schleifendurchläufen in die FIFO eingefügt.

Im folgenden Abschnitt wird die Realisierung der Basiskomponenten für die Kommunikation kurz präsentiert.

5.2.2.4 Anforderungsbezogene Auswahl der Kommunikationskomponenten und Darstellung eines Realisierungsbeispiels

Ein Großteil der dargestellten Mechanismen (Fehlermechanismen und PLC) sind in der zweiten Projektphase im Anschluss an die Erst-Entwicklung erfolgt, so dass für die Erstentwicklung der Architektur der Signal- und Datenverarbeitungseinheit folgende verfügbaren Komponenten ausgewählt wurden:

- Physical Layer: PLB
- Data Link Layer: DLLD

Die Realisierung der ausgewählten Kommunikationskomponenten und deren Verwendung werden anhand des einfachen Beispiels eines Senders und eines Empfängers demonstriert, welche auf unterschiedlichen FPGA-Modulen realisiert werden.

5.2.2.4.1 Realisierung eines einfachen Senders Innerhalb des Sender-FPGA-Moduls werden die Anwendungsdaten (21 uInt32-Werte) erzeugt. Diese werden anschließend zum Empfänger-FPGA-Modul übertragen und dort erfolgt die Weiterverarbeitung.

Das VI (Realisierung) des Sender-FPGA ist in Abbildung 5.2.15 dargestellt und gliedert sich in zwei Bereiche: auf der linken Seite befindet sich die Komponente zur physikalischen Übertragung der Daten und auf der rechten Seite die Anwendung. Die Kopplung beider Teilbereiche erfolgt über die FIFO *Send_uInt32*. Es finden sich drei Sub-VIs: die Nutzer-Anwendung, die Übertragung der einzelnen Datenelemente über die digitale Schnittstelle des FPGA-Moduls sowie die Eintragung der Arrayelemente in die FIFO zur Synchronisation der beiden Bereiche.

Das Sub-VI zum Kopieren der Nutzerdaten, welche als uInt32-Array vorliegen ist in Abbildung 5.2.16 dargestellt. Die einzelnen Datenwerte (uInt32) werden in die angegebene FIFO kopiert. Die zugehörige Gegenstelle ist in Abbildung 5.2.17 zu sehen. Hier werden abhängig vom gewählten Sendetakt Werte aus der FIFO entnommen.

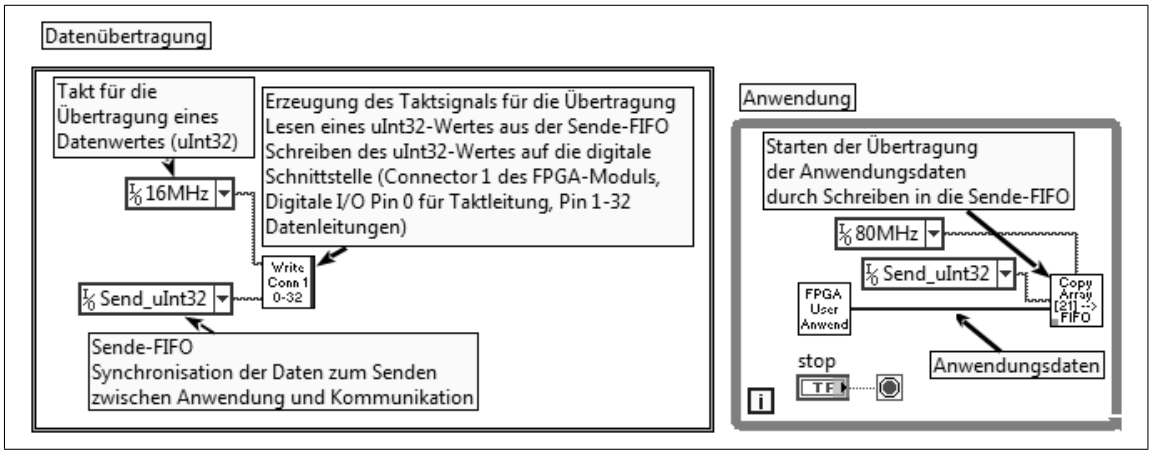


Abbildung 5.2.15: Realisierung einer einfachen FPGA-Anwendung mit Sendefunktion

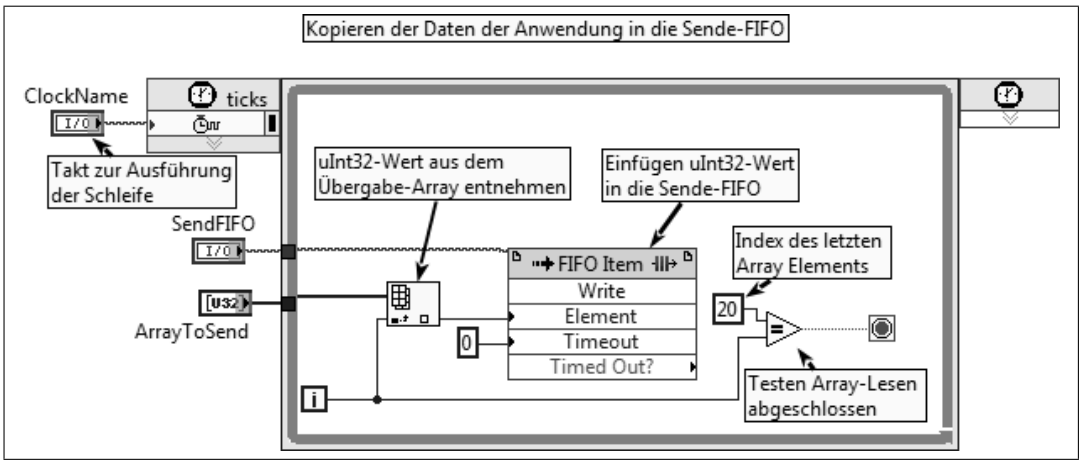


Abbildung 5.2.16: Sub-VI zum Kopieren eines Arrays mit 21 Elementen vom Typ uInt32 in eine FIFO zur Datenübertragung

Falls ein Wert vorhanden ist wird dieser auf die digitalen Outputs des FPGA-Moduls geschrieben und der Pegel der Taktleitung geändert. Über ein Kabel sind das sendende und das empfangende FPGA-Modul miteinander verbunden. Wurden die Daten auf die digitale Schnittstelle des Senders geschrieben können diese vom Empfänger gelesen bzw. ausgewertet werden.

5.2.2.4.2 Realisierung eines einfachen Empfängers Innerhalb des FPGA-Moduls zum Empfang werden zunächst die Daten von der digitalen Schnittstelle gelesen und nach erfolgreichem Empfang des gesamten Datensatzes (21 uInt32-Werte) weiterverarbeitet.

Das VI (Realisierung) des Empfänger-FPGAs ist in Abbildung 5.2.18 dargestellt und gliedert sich ebenso wie der Sender in zwei Bereiche: auf der linken Seite befindet sich die Komponente zur physikalischen Übertragung der Daten und auf der rechten Seite die Anwendung. Die Kopplung beider Teilbereiche erfolgt über die FIFO *Recei-*

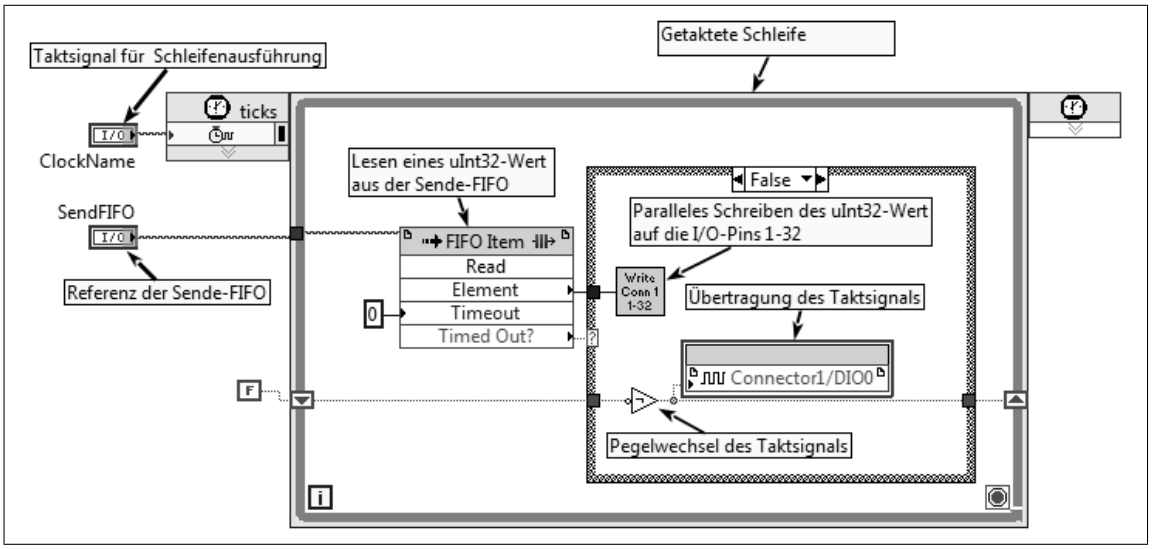


Abbildung 5.2.17: Sub-VI zum Lesen eines uInt32 Wertes aus einer FIFO und Übertragung über das digitale Interface

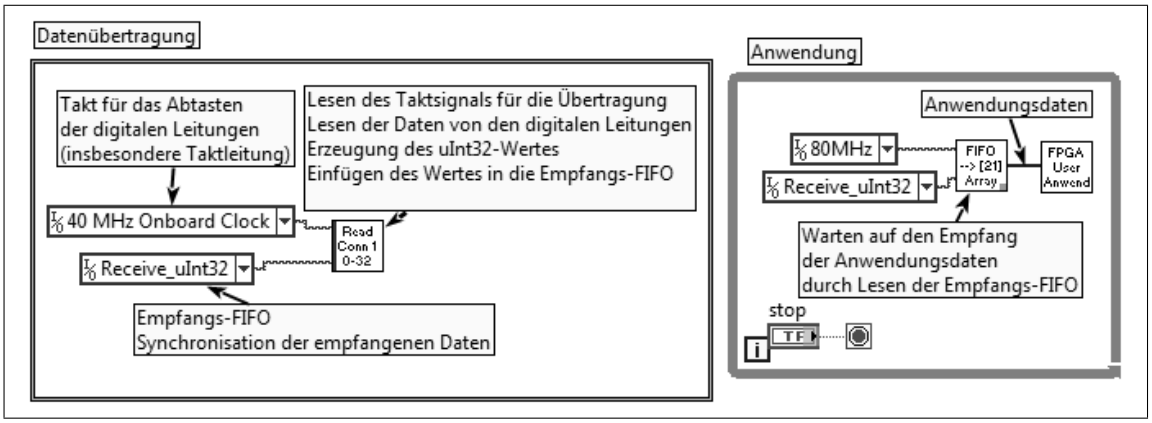
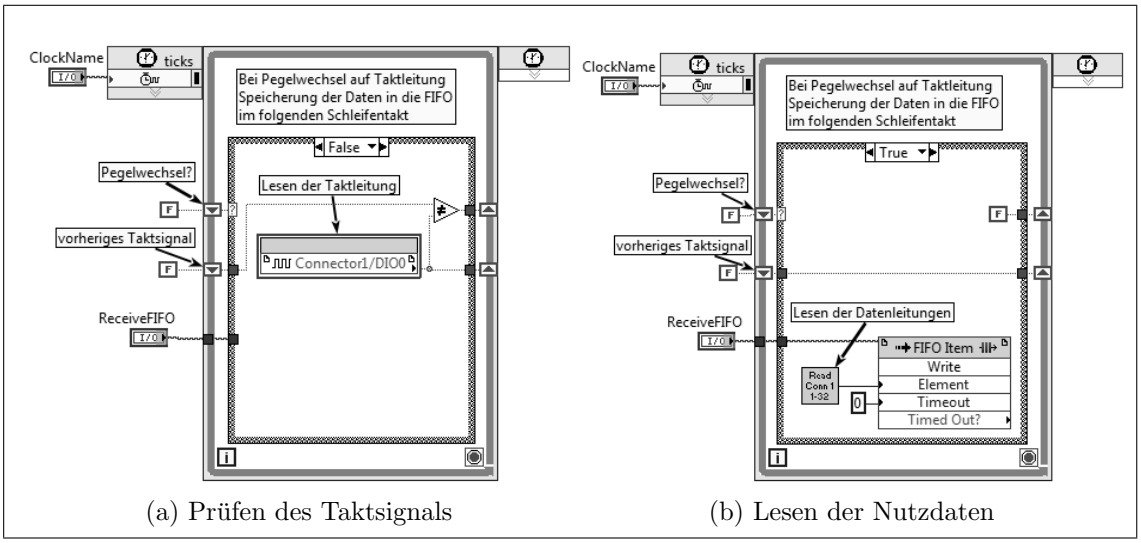


Abbildung 5.2.18: Realisierung einer FPGA-Anwendung mit Empfangsfunktion

ve_uInt32. Analog zum Sender finden sich drei Sub-VIs: die Nutzer-Anwendung, das Empfangsmodul sowie das Lesen aus der FIFO mit Erzeugung der Anwendungsdaten.

Das Empfangsmodul ist in Abbildung 5.2.19 dargestellt. Übergeben werden an das Sub-VI die Empfangs-FIFO sowie eine Clock zum Abtasten der Signale (mindestens zweifacher Sendetakt). Wird auf der Taktleitung ein Pegelwechsel detektiert, dies ist gleichbedeutend mit der Signalisierung der Übetragung von neuen Daten, werden im folgenden Empfangstakt die Datenleitungen gelesen, ein uInt32-Wert erzeugt und dieser anschließend in die Empfangs-FIFO geschrieben. Auf Seiten der Anwendung wird auf den vollständigen Datensatz gewartet und anschließend die Verarbeitung fortgeführt.

Die dargestellten Kommunikationskomponenten verwenden 32 Datenleitungen und sind lediglich ein Ausschnitt aus der realisierten Bibliothek für die Kommunikation zwischen zwei FPGA-Modulen. Es werden zusätzlich 4, 8 und 16 Datenleitungen für



(a) Prüfen des Taktsignals (b) Lesen der Nutzdaten

Abbildung 5.2.19: Sub-VI zum Lesen eines uInt32-Wertes von der digitalen Schnittstelle

die Kommunikation unterstützt. Zudem existiert eine Vielzahl von Komponenten für die Segmentierung und Desegmentierung von Daten (Kopieren eines Arrays in eine FIFO und umgekehrt).

5.2.2.4.3 Übersicht der entstandenen Kommunikationskomponenten Die Übertragung einer Datenstruktur zwischen zwei FPGA-Modulen erfolgt in sechs Schritten. Für jeden dieser einzelnen Schritte existieren mehrere unterschiedliche Komponenten (VIs). Dies ermöglicht eine flexible Komposition zur Realisierung einer konkreten Kommunikationsverbindung zwischen zwei FPGA-Modulen.

Der erste Schritt ist die Umwandlung der Datenstruktur in ein Array, dessen Datentyp ist abhängig von der Anzahl der verwendeten Datenleitungen. Anschließend werden die Array-Elemente in die FIFO kopiert, welche die Datenübergabe zur Sendeinstanz realisiert. Der Sendeteil wird abgeschlossen mit dem Auslesen der FIFO und Schreiben eines Datenwertes auf die digitalen Leitungen. Auf der Empfangsseite werden diese drei Schritte in umgekehrter Reihenfolge durchgeführt.

Für das Lesen von den digitalen Leitungen und das Schreiben von Werten auf die digitalen Leitungen existieren jeweils sechzehn in einer Bibliothek organisierte VIs. Aufgeteilt auf die vier Connectoren eines FPGA-Moduls gibt es jeweils vier Varianten, die sich durch die Anzahl der verwendeten Datenleitungen sowie die genutzten digitalen Leitungen unterscheiden: 32 Datenleitungen, 16 Datenleitungen unter Verwendung der digitalen Leitungen 0-16 (DIO 0-16), 16 Datenleitungen (DIO 17-33) und 4 Datenleitungen (DIO 35-39). Diese Kommunikationsbibliotheken für die Verarbeitungsschritte drei und vier sind anwendungsunabhängig.

Zusätzlich existieren für die anwendungsabhängigen Verarbeitungsschritte VIs für die unterschiedlichen Varianten (32-Bit, 16-Bit, 8- bzw. 4-Bit) zum Schreiben eines Arrays in eine FIFO (ArrayToFIFO) sowie zum Auslesen einer FIFO und Erstellen eines Arrays (FIFOtoArray). Ergänzt werden diese durch die Umwandlung von

konkreten Datenstrukturen in Arrays und umgekehrt sowie VIs zur Integration der Horizontalen und Vertikalen Kodierung in den Kommunikationsablauf.

5.2.2.5 Prototyping und Profiling von Kommunikationskomponenten

Analog zu dem zuvor dargestellten Entwurfsprozess ist, für eine modell- und simulationsbasierte Analyse dieses verteilten eingebetteten Systems, die Ermittlung der Charakteristiken der Kommunikationskomponenten erforderlich. Dies kann zum einen auf Basis der Spezifikation analytisch abgeschätzt (Operationen, Schleifen, Taktraten) oder im Rahmen einer Prototyping- und Profiling-Phase im Rahmen kleiner realistischer Szenarien ermittelt werden. Die so ermittelten Leistungsparameter bilden die Grundlage für die Parametrierung von Simulationsmodellen zur Unterstützung des Entwurfs und der Bewertung unterschiedlicher Architekturen und funktionalen Verteilungen für die Signal- und Datenverarbeitungseinheit der NPMM-200. Auf Grund des hohen Grades der Verteilung und der Parallelität kommt der Kommunikation innerhalb dieser Systemstruktur eine besondere Bedeutung zu und hat einen enormen Einfluss auf die Leistungsfähigkeit des Gesamtsystems.

5.2.3 Modellbasierte Analyse

Das Gesamtsystem der Signal- und Datenverarbeitungseinheit besteht aus einem sehr komplexen Gebilde von Hard- und Software-Komponenten. Bei der Systempartitionierung gibt es zunächst eine rein semantische Aufteilung in drei Subsysteme die als weitestgehend autonom betrachtet werden können. In den jeweiligen Systemen kommen diverse FPGA-Module zum Einsatz welche die Komplexität und den Entwicklungsaufwand des Gesamtsystems erhöhen. Ein Prototyping und Profiling, welches für kleine Teilkomponenten noch sinnvoll ist, wird auf der Gesamtsystem-Ebene sehr aufwändig. Der Einsatz von Modellen und Simulationen zur Untersuchung des Systems insbesondere des Zeitverhaltens ist somit angebracht.

5.2.3.1 Bezug zum vorgestellten Modellierungskonzept

Das vorgestellte Modellierungskonzept adressiert verteilte eingebettet Systeme im Segment Automotive. Bei einem Automobil handelt es sich um ein komplexes heterogenes Gesamtsystem welches eine heterogene Kommunikationsstruktur aufweist. Die Bussysteme erstrecken sich dabei über das ganze Automobil und verbinden die einzelnen Steuergeräte.

Diese Anwendungsdomäne unterscheidet sich vom betrachteten Anwendungsfall einer Signal- und Datenverarbeitungseinheit für eine Nanopositionier- und Nanomessmaschine deutlich. Im Wesentlichen gibt es nur eine zentrale Funktionalität in einem recht homogenen System mit einer deutlich geringeren Ausdehnung. Somit unterscheiden sich auch die Anforderungen an die Kommunikationsmedien. Es werden weitestgehend einfache Punkt-zu-Punkt-Verbindungen verwendet. Makroskopisch gesehen handelt es sich um ein einzelnes Steuergerät und keinen Verbund wie beim Automobil.

Bei einer etwas detaillierteren Analyse lassen sich die einzelnen Komponenten des Systems, insbesondere die FPGA-Module, als separate „Steuergeräte“ interpretieren. Zudem können die Punkt-zu-Punkt-Verbindungen als eine sehr einfache Buskommunikation mit nur 2 Teilnehmern gesehen werden. Somit ist eine Abbildung des grundlegenden Modellierungskonzeptes wie in Abbildung 5.2.20 dargestellt möglich. Hier wird ein Ausschnitt der Systemarchitektur der SDPU-NPMM200 dargestellt und auf diese der Modellierungsansatz angewendet. Dabei werden die Applikation (Host) und der Communication Controller jeweils auf dem FPGA-Modul realisiert. Da es sich lediglich um unidirektionale Punkt-zu-Punkt-Verbindungen handelt, kann ein FPGA-Modul mehrere CCs beinhalten zur Kommunikation mit unterschiedlichen Teilnehmern.

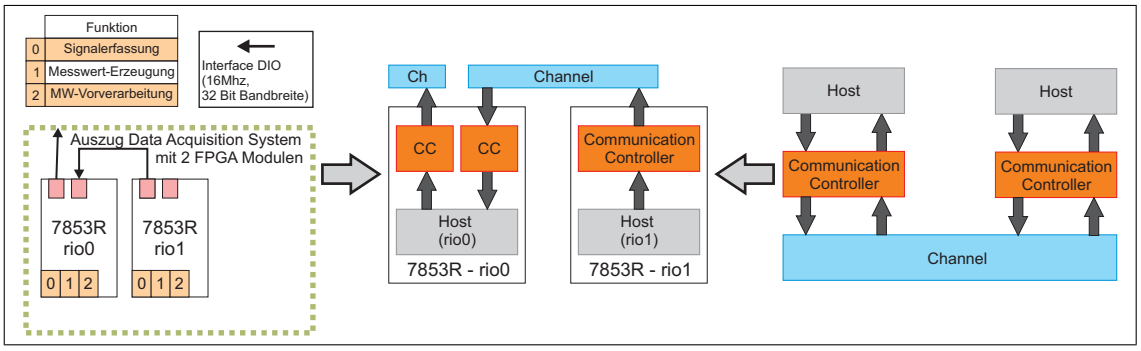


Abbildung 5.2.20: Anwendung des Modellierungskonzeptes auf die Signal- und Datenverarbeitungseinheit der NPMM200

Die nachfolgenden entwickelten Modelle sowie die durchgeführten Prototyping und Profiling Aufgaben sind Grundlage für die angeschlossenen modellbasierten Untersuchungen in Hinblick auf eine leistungsfähige Architektur für eine Signal- und Datenverarbeitungseinheit.

5.2.3.2 Modellbibliothek für die Kommunikation

Die Modelle zur Verifikation der Protokolle an sich werden nicht betrachtet, da lediglich die Kommunikationskomponenten zur Modellierung der Systemarchitektur der Signal- und Datenverarbeitung von Relevanz sind und nicht Protokollentwicklung im engeren Sinne.

Die Modelle abstrahieren von der realen physikalischen Verbindung, die physikalischen parallelen Leitungen werden daher nicht berücksichtigt. Zudem wird auf die explizite Modellierung des Taktsignals verzichtet. Modelliert wird ein Übertragungskanal, dessen Datenbreite der Anzahl der physikalischen parallelen Datenleitungen entspricht. Das Taktsignal wird durch den Event beim Datenempfang repräsentiert.

5.2.3.2.1 Abbildung der definierten Protokolle in die Modellstruktur Für die in Kapitel 5.2.2 vorgestellten Protokollfunktionen müssen zur Modellierung des Gesamtsystemverhaltens geeignete Module in einer Modellbibliothek zusammengefasst

werden. Zu diesem Zweck erfolgt eine weiterführende Abbildung der Kommunikationsprotokolle in das Modellierungskonzept. Es wird sich auf die Echtzeit-Kommunikation zwischen zwei FPGA-Modulen beschränkt, der Datenaustausch über die PXI-Backplane (PCI-Bus) mit den niedrigeren Zeitanforderungen wird nicht betrachtet.

Eine Besonderheit bei der verwendeten Hardware ist, dass sich Protokollfunktionen und Anwendungsfunktionen gemeinsam auf einem FPGA befinden, dies ist gleichbedeutend mit einem integrierten Communication-Controller. Diese Eigenschaft hat keine direkte Auswirkung auf die Modellierung. Es werden entsprechend die drei Komponenten Host, Communication Controller und Channel unterschieden.

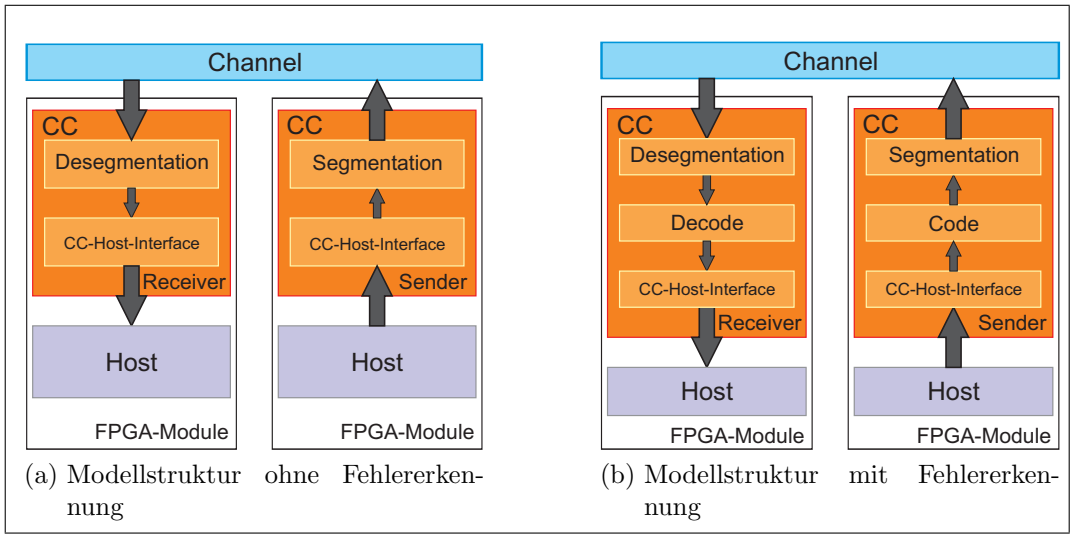
5.2.3.2.2 Host Der Host beinhaltet die Applikationsfunktionen, im konkreten Anwendungsfall sind dies Erfassung von analogen Signalen, Signalverarbeitung, Korrekturalgorithmen und Regelungsalgorithmen. Im Speziellen wird innerhalb der Applikation für die Berechnung von Algorithmen zusätzlich ein Softcore-Prozessor (Pacholik, Klöckner, Müller, Gushchina & Fengler, 2011) eingesetzt. Für die abstrakte Modellierung der Systemfunktionalitäten sind Zeitverzögerungen und die Erzeugung von Datenstrukturen ausreichend. Die Modellierung auf Applikationsebene ist zunächst nicht Gegenstand der Betrachtung.

5.2.3.2.3 Channel Der Channel spiegelt die physikalische Verbindung zwischen den jeweiligen FPGA-Modulen wieder. Diese besitzt zwei wesentliche Eigenschaften, dies sind die Anzahl der parallelen Daten-Leitungen sowie die Taktrate der Clock-Leitung. Aus diesen Parametern ergibt sich in Verbindung mit der Größe der Datenpakete die Verzögerung auf dem Kommunikationsmedium. Das Medium besteht dabei aus einem speziellen geschirmten Verbindungskabel zwischen den beiden FPGA-Modulen. Spezielle Arbitrierungsmechanismen auf Leitungsebene, ähnlich wie beim CAN-Bus, gibt es durch den unidirektionalen Datenaustausch zwischen zwei Teilnehmern nicht.

5.2.3.2.4 Communication Controller Durch den sehr einfachen Aufbau des Kommunikationsprotokolls entfallen aufwändige Zugriffsmechanismen. Jeder Teilnehmer kann nur eine Rolle in der Kommunikation einnehmen, Sender oder Empfänger. Somit kann auf Modellebene diese Aufteilung analog erfolgen.

Die Funktionen für die einfachste Form der Datenübertragung beschränken sich auf die Umwandlung der Datenstrukturen der Applikationsebene in Pakete für die Kommunikation (Segmentierung und Desegmentierung). Abhängig von der Ausbaustufe können weitere Funktionen integriert werden, durch einfache Verringerung des Abstraktionsgrades (detailliertere Modellierung). Die Erweiterungen beziehen sich insbesondere auf Mechanismen zur Fehlererkennung und die damit verbundene Codierung und Decodierung von Nachrichten.

Durch die Mechanismen zur Fehlererkennung entstehen zusätzliche Verzögerungen. Diese entstehen durch die Codierung und Decodierung sowie durch die Übertragung zusätzlicher Daten (Kontrolldaten). Ein Beispiel für den Aufbau eines Modells ist in Abbildung 5.2.21 dargestellt, es wird eine Variante mit und ohne Fehlererkennung dargestellt. Durch die Austauschbarkeit der einzelnen Funktionen lässt sich das konkrete Kommunikationsprotokoll flexibel realisieren.



(a) Modellstruktur ohne Fehlererkennung

(b) Modellstruktur mit Fehlererkennung

Abbildung 5.2.21: Modellaufbau als Beispiel

Die entstandene Modell-Bibliothek bildet eine Grundlage für die Untersuchungen unterschiedlicher Systemvarianten für die Signal- und Datenverarbeitungseinheit der NPMM-200.

5.2.3.3 Abstraktionsebenen der Kommunikationsmodellierung

Für die Modellierung der Kommunikation bei diesem recht einfachen Protokoll (ohne Betrachtung von zusätzlichen Mechanismen wie Fehlerkorrektur) können zwei unterschiedliche Ebenen der Abstraktion betrachtet werden: Paketorientierung und Bit-Orientierung.

5.2.3.3.1 Paketorientierung In frühen Phasen der Entwicklung sowie für Entscheidungen auf Anwendungs- und Systemebene ist eine paket-orientierte Modellierung geeignet. Der reduzierte Detaillierungsgrad und die damit verbundene geringe Simulationszeit sind hier von Vorteil. Auch für Aussagen über die Gesamtsystem-Performance sind die paket-orientierten Modelle ausreichend.

In den paket-orientierten Modellen werden bei der Kommunikation komplexe Datenpakete zwischen Kommunikationsteilnehmern ausgetauscht, somit erfolgt eine Abstraktion von der Bitübertragungsschicht. Folglich werden bitbezogene Mechanismen nicht berücksichtigt.

5.2.3.3.2 Bit-Orientierung Die Modelle auf Bitlevel ermöglichen detaillierte Timing-Analysen. Dies dient insbesondere zur Optimierung von Komponenten. Dabei können Varianten von Modellkomponenten direkt miteinander verglichen werden, um so die Komplexität von Modellen und damit auch die Simulationszeit auf ein überschaubares Maß zu reduzieren.

Insbesondere bei der Verwendung von FPGA-Modulen bieten sich weitere Vorteile. Auf einem hohen Detaillevel können unterschiedliche Varianten der Realisierung verglichen werden. Implementierungsvarianten können sehr detailliert modelliert werden und so das zeitliche Verhalten, sogar auf FPGA-Takt-Level, simuliert werden.

Hierdurch sind Performance und Timing-Aussagen, insbesondere Vorhersagen möglich ohne die Notwendigkeit einer aufwändigen Prototyping-Phase.

Bei den verwendeten Kommunikationsprotokollen kommt dies insbesondere bei funktionalen Erweiterungen zum Tragen. Die unterschiedlichen Mechanismen zur Fehlerbehebung und Fehlererkennung können auf diese Art und Weise bewertet werden.

5.2.4 Systemarchitektur

Die Prüfung der Architekturentscheidungen erfolgt jeweils parallel mit Entwicklung und Realisierung der spezifischen Anwendungsalgorithmen, dabei erhöht sich jeweils der Grad der Detailierung des Systemmodells. Die Analyse der Leistungsfähigkeit der Systemarchitektur basiert auf simulativen Untersuchungen.

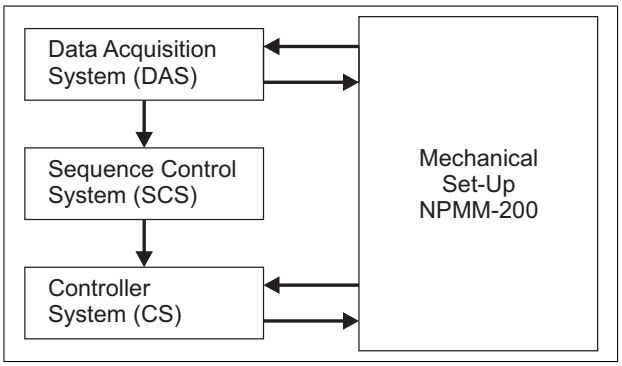


Abbildung 5.2.22: Grundsäztlicher Aufbau der Signal- und Datenverarbeitungseinheit der NPMM200

In der komplexen Signal- und Datenverarbeitungseinheit (SDPU) wird anhand der drei Hauptfunktionsbereiche (Datenerfassung, Messablauf, Antriebssteuerung), wie in Abbildung 5.2.22 dargestellt, eine Systemaufteilung in drei separate Teilsysteme vorgenommen. Durch die logische und funktionale Partitionierung entstehen drei Teilsysteme: das Signal- und Datenerfassungssystem (DAS), das System zur Ablaufsteuerung (SCS) und das Regelungssystem (CS). Diese Systeme basieren jeweils auf einer Kombination von PXI- und FPGA-Technologie.

Die Signalerfassung und Vorverarbeitung erfolgt innerhalb des DAS ebenso wie die Messdaten-Nachverarbeitung, zu der komplexe Berechnungen und Messwertkorrekturen gehören. Die Messdaten werden vom DAS zum SCS übertragen. An dieser Stelle wird die Ablaufsteuerung durchgeführt und die Sollwerte und weitere Steuergrößen werden erzeugt und zum CS weitergeleitet. Innerhalb des CS erfolgen die Regelung und die Ausgabe der Antriebssignale zum mechanischen System.

Der hohe Grad der Flexibilität, Modularität und Parallelität insbesondere durch die verfügbaren FPGA-Module führt zu einem heterogenen System. Bei diesem sollen die Modelle für simulationsbasierte Untersuchungen zur Bewertung von Architekturvarianten sowie Kommunikationsinfrastrukturen eingesetzt werden.

In der ersten Projektphase ist der Wissensstand sehr gering dies betrifft insbesondere den mechanischen Aufbau, die Anforderungen an das System und die Leistungs-

fähigkeit der Hardwarekomponenten. Aus diesem Grund sind vielfältige Untersuchungen zur Unterstützung von Entwurfsentscheidungen notwendig.

Nachfolgend erfolgt die kurze Vorstellung ausgewählter Analysen zur Darstellung der Anwendung der Modelle in unterschiedlichen Abstraktionsstufen während des Entwurfsprozesses.

5.2.4.1 Grundlegende Partitionierungsentscheidungen - Kommunikationsmechanismen

Sehr abstrakte paket-orientierte Modelle, die eine reine Verzögerung der Daten auf der Kommunikationsschnittstelle vornehmen, werden zur Untersuchung grundsätzlicher Varianten für die Systemarchitektur genutzt.

Als Beispiel dieser Untersuchungen wird nachfolgend das Teilsystem CS (Controller System) betrachtet, welches das Regelungssystem enthält. Hier werden drei unterschiedliche Architektur- und Technologie-Varianten miteinander verglichen, welche sich durch den Ausführungsort der Berechnung des Regelungsalgorithmus unterscheiden:

CS Variante 1 Berechnung der Regelungsalgorithmen auf dem RT-Controller, Ausgabe der Stellgrößen nach Achsen aufgeteilt über die analogen Schnittstellen der FPGA-Module

CS Variante 2 Berechnung der Regelungsalgorithmen auf zwei zusätzlichen DSP-Modulen (XY-Regelung, Z-Regelung), Ausgabe der Stellgrößen über die analogen Schnittstellen der FPGA-Module

CS Variante 3 Berechnung der Regelungsalgorithmen und Ausgabe der Stellgröße innerhalb der FPGA-Module (Aufteilung XY-Regelung/Z-Regelung)

Die drei unterschiedlichen Architektur-Varianten für dieses Teilsystem sind in Abbildung 5.2.23 zu sehen. Die einzelnen Komponenten (FPGA-Module und RT-Controller) sind als Blöcke dargestellt und die Kommunikationspfade werden durch Pfeile repräsentiert. Zusätzlich sind die einzelnen Funktionen und ihr jeweiliger Ausführungsort durch Nummern gekennzeichnet.

Die Simulationsergebnisse dieser Architekturstudie in einer sehr frühen Entwurfsphase verdeutlichten einen Vorteil bei Verwendung der dritten Architektur-Variante (CS Variante 3). Die Darstellung der Modelle sowie der Simulationsergebnisse folgt in Abschnitt 5.2.4.3. Die begleitend durchgeführten Prototyping und Profiling-Resultate unterstützen diese grundsätzliche Architekturentscheidung (Müller et al., 2013).

5.2.4.2 Mechanismen zur Datenfusion

Bei der entwurfsbegleitenden Modellierung, Simulation und Analyse zur Unterstützung von Entwurfsentscheidungen werden die Analysen mit fortschreitender Entwicklungsphase detaillierter. Neben der funktionalen Verteilung spielt auch die Auswahl

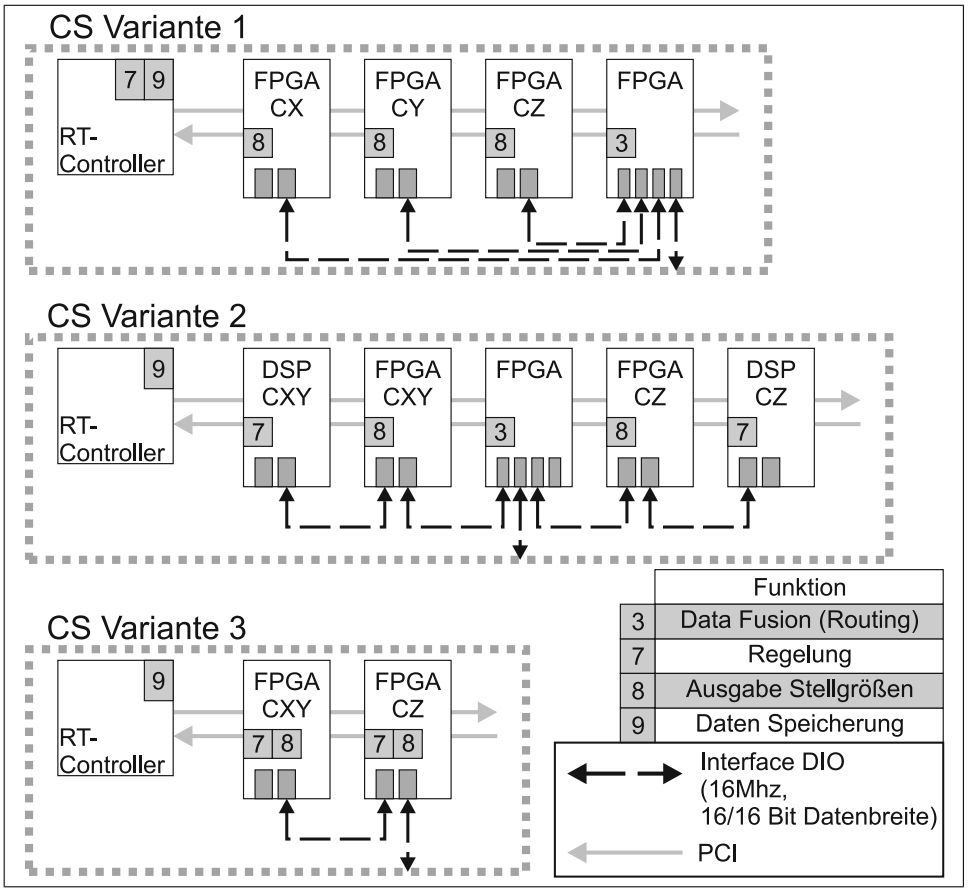


Abbildung 5.2.23: Architekturvarianten zur Realisierung des Teilsystems CS

geeigneter Mechanismen zur Realisierung eine Rolle. Eine zentrale Aufgabe hinsichtlich der Kommunikation innerhalb des Systems ist das Zusammenführen von parallelen Datenpfaden für eine folgende Weiterverarbeitung.

In Abbildung 5.2.24 ist ein Auszug des Teilsystems DAS zu sehen, welches eines der Systemteile darstellt in dem eine Fusion von Messdaten notwendig ist. Die vorverarbeiteten Messdaten werden von den FPGA-Modulen *Quelle1*, *Quelle2* und *Quelle3* zu dem FPGA-Modul *Fusion*, welches die Datenfusion übernimmt gesendet, und von dort zum FPGA-Modul *Ziel* weitergeleitet, welches die Weiterverarbeitung der fusionierten Messdaten übernimmt. Zwei Szenarien werden dabei separat betrachtet, erstens die FPGA-Module senden Pakete identischer Größe und zweitens die Pakete dürfen unterschiedliche Größen besitzen.

In einem folgenden Entwicklungsschritt wird die Modellbibliothek für die Untersuchung unterschiedlicher Mechanismen zur Datenfusion verwendet. In Abbildung 5.2.24 werden zwei Varianten für die Datenfusion dargestellt. Die dritte Variante ist eine Mischung aus den ersten beiden Varianten. Bei den Varianten erfolgt jeweils eine direkte Weiterleitung eines jeden Datenpaketes, dadurch kommt es zu einer Vermeidung von Verzögerungen, welche durch das Abwarten des Empfangs eines kompletten Datensatzes entstehen würden.

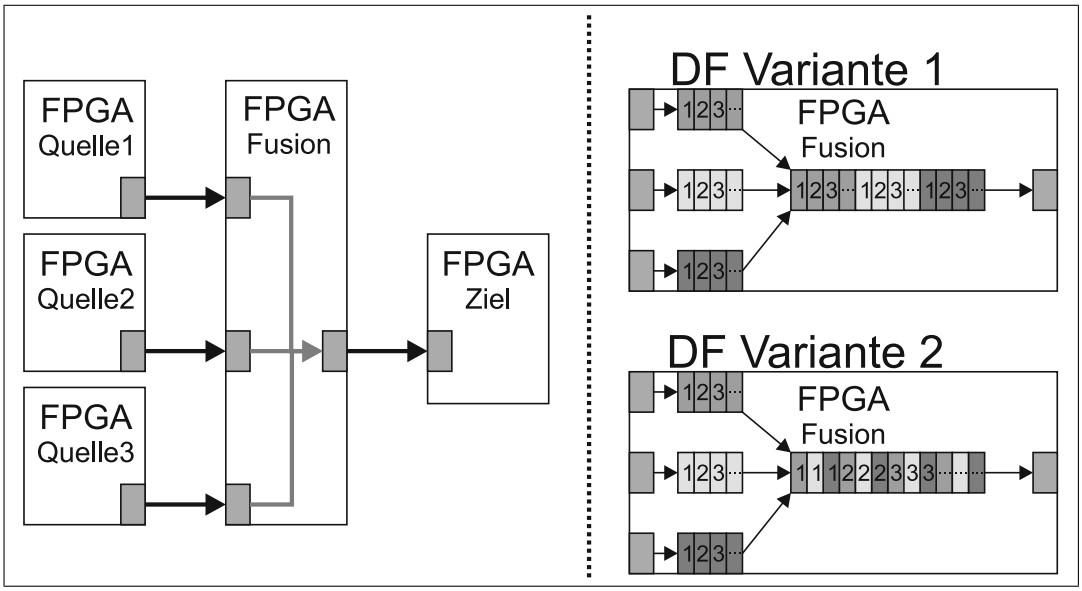


Abbildung 5.2.24: Kommunikationsstruktur und Varianten zur Datenfusion

DF Variante 1 Bei der ersten Variante werden die einzelnen FPGA-Module nacheinander abgearbeitet. Es wird jeweils eine feste Anzahl von Datenpaketen eines jeden Senders weitergeleitet.

DF Variante 2 Die zweite Variante arbeitet nach einem Round-Robin-Verfahren, die einzelnen Daten werden abwechselnd nacheinander abgefragt und weitergeleitet. Voraussetzung an dieser Stelle ist, dass alle Datensätze die gleiche Größe besitzen.

DF Variante 3 Die dritte Variante ist eine sehr flexible Abwandlung der ersten beiden Varianten. Die Empfangsqueues werden jeweils abgearbeitet bis keine Daten mehr vorhanden sind, danach erfolgt die Abarbeitung der nächsten Empfangsqueue. Eine Festlegung der Anzahl oder eine Beschränkung auf Datensätze mit gleicher Größe ist nicht erforderlich. Der Physical Layer der Kommunikationsschnittstelle wird erweitert, um über drei zusätzliche Signalleitungen die Herkunft der Daten zu kennzeichnen.

In dem konkreten Anwendungsfall hat sich in den Simulationen gezeigt, dass auf Grund der spezifischen Anforderungen die sehr einfache erste Variante als geeignete Version für die Realisierung ausgewählt werden kann. In einem abschließenden Prototyping konnten die Ergebnisse aus der Simulation nachgewiesen werden. Die prototypische Untersuchung der verschiedenen Varianten erfolgte im Rahmen der betreuten Diplomarbeit von Boller (2010).

5.2.4.3 Modellbasierte Untersuchung der Systemarchitektur

Die Modelle und die Simulationsergebnisse wurden am Beispiel der Untersuchungen zu den Partitionierungsentscheidungen wie sie in Abschnitt 5.2.4.1 dargestellt sind demonstriert. Die drei Varianten zur Realisierung des Regelungssystems werden mit

Hilfe von geeigneten Modellen anhand ihrer Performance-Eigenschaften miteinander verglichen. Die praktischen Untersuchungen erfolgten im Rahmen der betreuten Bachelorarbeit von Rauh (2009).

Wichtige Randbedingungen dieser Analysen sind:

- Man befindet sich in einer frühen Entwurfsphase .
- Es gibt nur eingeschränktes Wissen über die konkreten Realisierungen von Komponenten sowie deren Performance-Eigenschaften.
- Es gibt kein oder nur ein sehr eingeschränktes Expertenwissen zu den verfügbaren Hardware-Komponenten sowie deren Performance-Eigenschaften.

Aufgrund dieser Randbedingungen werden sehr abstrakte Verarbeitungsmodelle verwendet. Die Verlässlichkeit der Ergebnisse ist durch die große Anzahl von nicht verifizierten Annahmen und Parametern eingeschränkt, im Verlauf der Entwicklung müssen die Analysen entwurfsbegleitend ggf. wiederholt und geprüft werden, wobei der Detaillierungsgrad erhöht werden kann und Informationen aus dem Prototyping einfließen können.

5.2.4.3.1 Aufbau des Systemmodells Die oberste Ebene des Simulationsmodells spiegelt direkt die Systemarchitektur wieder. In Abbildung 5.2.25 sind daher die Hauptelemente der Signal- und Datenverarbeitungseinheit Data Acquisition System (DAS), Sequence Controller System (SCS) und Controller System (CS), welche jeweils ein PXI-System widerspiegeln, sowie die Maschine zu finden. Der Signal- und Datenaustausch zwischen den einzelnen Komponenten erfolgt über die Modul-Ports. Die detaillierte interne Struktur wird an dieser Stelle nicht ausführlich betrachtet, exemplarisch wird lediglich eine Variante des Controller Systems (Abbildung 5.2.26) kurz erläutert. Die unterschiedlichen System-Modelle für die verschiedenen Varianten unterscheiden sich lediglich durch das Modell des Controller Systems.

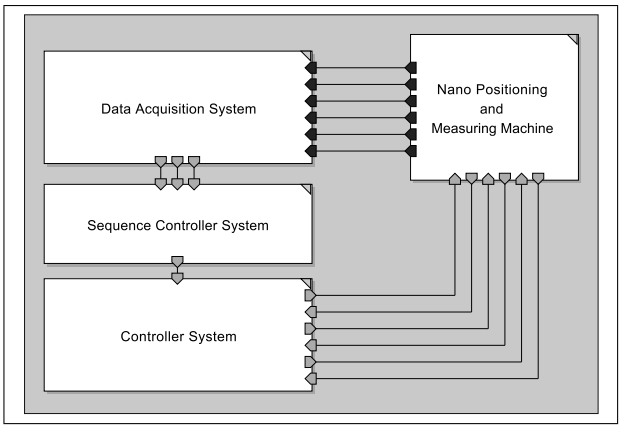


Abbildung 5.2.25: Modell der obersten Systemebene

Das in Abbildung 5.2.26 dargestellte Modul Controller System entspricht im realen Hardware-Aufbau einem PXI-System mit diversen FPGA-Modulen. Die zur Verarbei-

tung verwendbaren Komponenten (FPGA-Module und RT-Controller) sind gleichfarbig. Die einzelnen Verarbeitungskomponenten haben dabei unterschiedliche Schnittstellen, die zum Datenaustausch verwendet werden können. Hierbei handelt es sich um die digitalen und analogen Schnittstellen der FPGA-Module sowie den PCI-Bus, der die einzelnen Hardware-Module in einem PXI-Chassis verbindet. Die Module der Modelle für die Komponenten zur internen Kommunikation sind dunkel eingefärbt. Dies ist ein Modul für den PCI-Bus sowie insgesamt sechs Module für den Datenaustausch zwischen den FPGA-Modulen, dies ist ersichtlich anhand der in Abschnitt 5.2.2 vorgestellten Kommunikationsmechanismen. Die Ports des Regelungs-Moduls bilden die Schnittstelle zur Maschine, hierbei handelt es sich um analoge Eingangssignale zur Erfassung der Ist-Ströme der Motoren (MS1, MS2, MS3) sowie analoge Ausgangssignale zur Ansteuerung der Motoren bzw. Ausgabe der Stellgrößen (Sollströme SG1, SG2, SG3), und zum SCS (IW_SW), welches die Ist- und Sollwerte sowie weitere Steuerinformationen zum Regelungssystem überträgt.

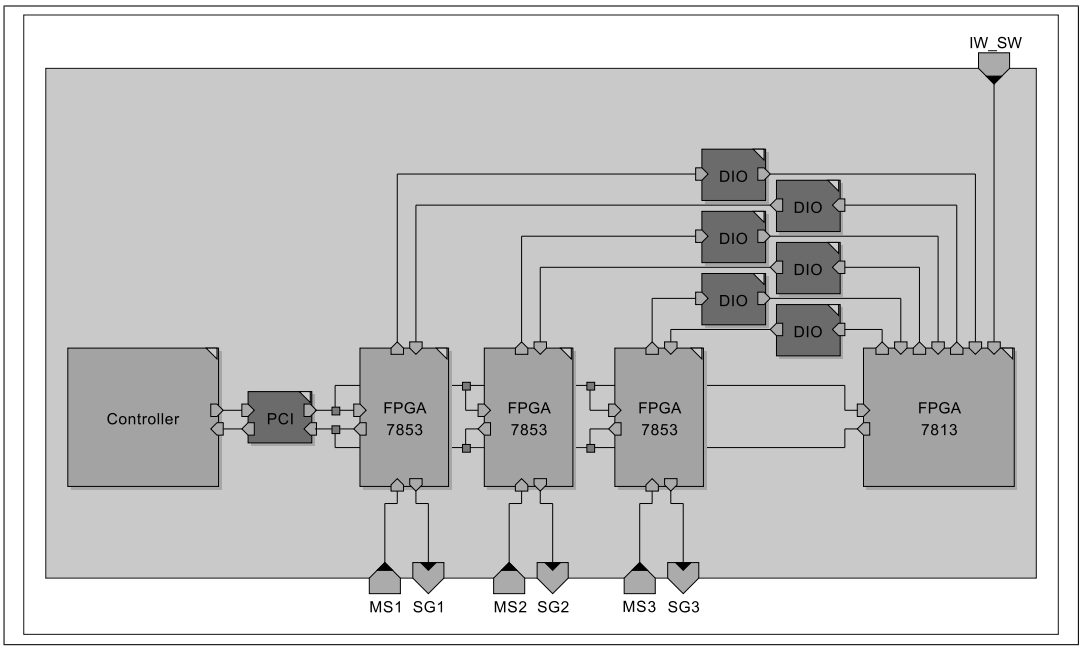


Abbildung 5.2.26: Modell der *CS Variante 1* des Subsystems Regelung

Die wesentlichen Parameter der verarbeitenden Module sind die Ausführungsdauer der Funktionen und die Größe der Datenstrukturen die verarbeitet und zur Weiterverarbeitung anderen Funktionen zur Verfügung gestellt werden müssen. Für die Kommunikation über die digitale Schnittstelle, bei der es sich um eine unidirektionale Punkt-zu-Punkt Verbindung zwischen zwei Teilnehmern (einem Sender und einem Empfänger) handelt, ist ausschließlich die Verzögerung bei der Übertragung in den ersten Analysen relevant. Diese ist abhängig von drei Parametern der Übertragungsrate, der Anzahl der verfügbaren parallelen Leitungen sowie der Paketgröße.

5.2.4.3.2 Verarbeitung innerhalb des Modells Um die unterschiedlichen CS Varianten zu vergleichen werden im Gesamtsystem vier Verarbeitungsphasen (Phase 1

bis Phase 4) definiert, welche teilweise noch weiter unterteilt werden.

Phase 1 Die erste Phase umfasst die Messwerterfassung und gliedert sich in drei parallele Teilbereiche. In Teilbereich A wird die Erfassung und Verarbeitung der Signale der Referenzsensoren durchgeführt. Die Erfassung und Verarbeitung der Signale des Tastsystems wird in Teilbereich B berücksichtigt. Der dritte Teilbereich C umfasst die Signalerfassung und Verarbeitung des Positions- und Winkelmesssystems. Nach Abschluss aller drei Teilbereiche kann die folgende Phase beginnen. Die Phase 1 ist dem Systemteil DAS zugeordnet.

Phase 2 Die Phase 2 berücksichtigt die Verarbeitung innerhalb des Systemteils SCS, welcher sich für die Ablaufsteuerung und Bahnplanung verantwortlich zeigt. Dabei können zwei Teilbereiche unterschieden werden: Teilbereich A mit Kommunikationsaufgaben (Empfang und Verteilung der Messwerte aus Systemteil DAS) und Teilbereich B, der die Funktionalitäten der Ablaufsteuerung und der Bahnplanung beinhaltet sowie die Weiterleitung der erzeugten Sollwerte zum Regelungssystem übernimmt.

Phase 3 Die Phasen 3 und 4 lassen sich dem Systemteil CS zuordnen. In der Phase 3 erfolgt die Berechnung der Regelung.

Phase 4 In der abschließenden Phase 4 wird die Verteilung der berechneten Stellgrößen durchgeführt.

5.2.4.3.3 Parametrierung des Systemmodells Die Verarbeitungszeiten von einigen internen Funktionen, insbesondere der Regelungsalgorithmen, wurden in dieser frühen Phase noch nicht durch ein Prototyping auf realen Hardwarekomponenten bestimmt. Der Vergleich bezieht sich daher im Wesentlichen auf die Kommunikationsarchitektur.

Die Verarbeitung innerhalb des DAS konnte durch Prototyping und Profiling bereits bestimmt werden und das Modell wird entsprechend parametriert. Für die Berechnung der Regelungsalgorithmen wird unabhängig von der konkreten CS Variante eine Dauer von 30 μs festgelegt, da differenzierte Prototyping Informationen nicht zur Verfügung stehen. Die Parametrierung der Kommunikation erfolgt durch die Festlegung der Größen der Datenstrukturen in den verarbeitenden Modulen, sowie die Einstellung der Datenübertragungsrate (16 MHz) und die Festlegung der Anzahl der Kanäle (32 Kanäle für die Realisierung der unidirektionalen Kommunikation und 16 Kanäle für die Realisierung der bidirektionalen Kommunikation).

5.2.4.3.4 Simulationsergebnisse Der Vergleich der drei unterschiedlichen Varianten des Controller Systems erfolgt auf Basis der definierten vier Phasen. In Abbildung 5.2.27 sind die Resultate dargestellt. Es wird deutlich, dass die Varianten 2 und 3 Vorteile haben, da bei beiden die zeitintensive Datenkommunikation über den PCI-Bus entfällt. Die Kommunikation hat einen erheblichen Einfluss auf die Gesamtverzögerung. Die Unterschiede im zeitlichen Verhalten zwischen der Variante 2 und

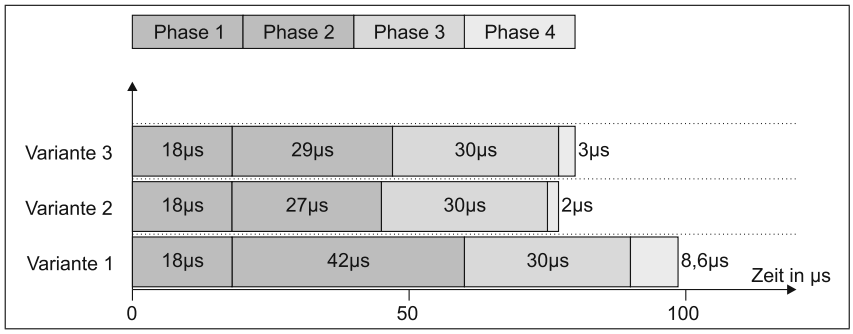


Abbildung 5.2.27: Darstellung der Ergebnisse der Simulation

3 sind nur marginal. Aufgrund des geringeren Hardware-Aufwandes hat Variante 2 einen Vorteil.

Im weiteren Verlauf der Entwicklung muss, sobald weitere verifizierte Leistungsparameter zur Verfügung stehen, die Auswahl geprüft werden, da teilweise die Verarbeitungszeiten (insbesondere innerhalb des Regelungssystem) nicht auf Profiling-Daten basieren.

5.3 Analyse - Optimierung und Strukturanpassung

Im Folgenden wird exemplarisch und auszugsweise die Anwendung der definierten und erstellten Modellbibliotheken im Rahmen der zweiten Projektphase mit dem Schwerpunkt einer Systemanalyse und Optimierung demonstriert.

In einem ersten Schritt erfolgt die Analyse des Zeitverhaltens und der zeitlichen Eigenschaften des Prototypen der Signal- und Datenverarbeitungseinheit. Ausgehend von dieser werden Möglichkeiten zur Optimierung und Anpassungen der Struktur betrachtet. Zunächst werden drei Architektur-Varianten aufgezeigt, die durch systembezogene Optimierungen und Strukturanpassungen entstehen. Die einzelnen Varianten bedingen dabei Änderungen im Bereich der Gesamtsystem-Anforderungen. Ableiten lassen sich variable nutzerspezifische Architekturen des Signal- und Datenverarbeitungssystems (SDPU) anhand der Anforderungen. Diese können im Hinblick auf Variationen im mechanischen Aufbau oder auf Basis von Nutzeranforderungen im Sinne von unterschiedlichen Ausbau- und Leistungsstufen der SDPU entstehen (zum Beispiel ein Minimalsystem bezüglich der Kosten).

Im folgenden Schritt wird der Einsatz der kommunikationsbezogenen Modellbibliotheken diskutiert und abschließend Aspekte von unterschiedlichen Abstraktionsebenen aufgezeigt.

5.3.1 Definition von Architekturvarianten

Exemplarisch werden an dieser Stelle ausgewählte Architekturvarianten mit ihren spezifischen Eigenschaften kurz präsentiert. Es handelt sich hier lediglich um eine Auswahl möglicher Systemarchitekturen. Ausgangspunkt für die Anpassungen und Variationen bildet der Prototyp der Signal- und Datenverarbeitungseinheit. Die betrachteten Varianten berücksichtigen insbesondere folgende Aspekte:

- Architekturanpassungen zur Verbesserung der Leistungsfähigkeit.
- Architekturanpassung zur Reduzierung der Komplexität unter Tolerierung einer verminderten Leistungsfähigkeit.
- Architekturanpassung zur Reduzierung der Komplexität ohne Einschränkung der funktionalen Leistungsfähigkeit.
- Systeminterne Variationen im Hinblick auf die interne Kommunikation.

5.3.1.1 Architekturvariante des Prototypen (Architekturvariante Basic)

Die Architekturvariante Basic stellt die grundlegende Systemarchitektur der Signal- und Datenverarbeitungseinheit (SDPU) der NPMM-200 dar, wie sie auch in (Percle, Klöckner, Manske & Fengler, 2011) und (Balzer, Klöckner, Zschäck & Percle, 2012) dargestellt wird.

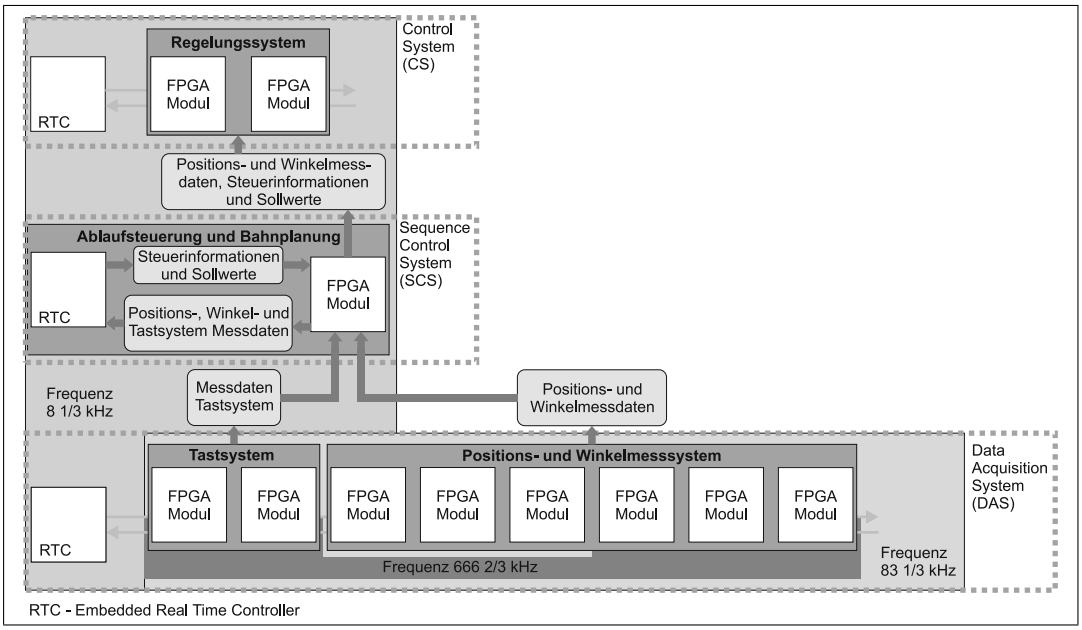


Abbildung 5.3.1: Grundarchitektur der Signal- und Datenverarbeitungseinheit

Die Struktur dieser SDPU ist in Abbildung 5.3.1 dargestellt und unterteilt sich in vier funktionale Teilbereiche (Regelungssystem, Ablaufsteuerung und Bahnplanung, Tastsystem sowie Positions- und Winkelmesssystem). Diese funktionalen Teilbereiche werden wiederum auf drei Systemkomponenten (DAS, SCS, CS) zur Signalerfassung und Datenvorverarbeitung, zur Ablaufsteuerung und Bahnplanung sowie zur Regelung aufgeteilt. Kennzeichnend für diese Systemarchitektur ist, dass die Echtzeitverarbeitung, welche den Bereichen Signalerfassung, Datenvorverarbeitung sowie Regelung zugeordnet werden kann, innerhalb der FPGA-Module erfolgt. Dabei wird der zeitkritische Datenaustausch zwischen den FPGA-Modulen über die in den vorherigen Abschnitten vorgestellten Kommunikationsmechanismen realisiert. Neben den

FPGA-Modulen wird für die maschinennahe Datenverarbeitung und Steuerung einer der drei vorhandenen Realtime-Controller eingesetzt, der die Ablaufsteuerung, Systemüberwachung und Bahnplanung durchführt.

Dieser Systemaufbau ist der Ausgangspunkt für Überlegungen und Untersuchungen zur Optimierung von Architektur, Struktur, funktionaler Verteilung und der Kommunikation unter applikations- und systemspezifischen Gesichtspunkten. Das Zeitverhalten des Prototypen ist der Abbildung A.7.1 in Anhang zu entnehmen.

5.3.1.2 Architekturvariante FPGA-basierte Echtzeit-Verarbeitung

Die erste Variation der Basis-Architektur wird unter dem Gesichtspunkt der Steigerung der Performance-Eigenschaften insbesondere der Verbesserung des zeitlichen Verhaltens betrachtet. Diese ist motiviert durch die Untersuchungen in (Zschäck et al., 2013) und (Zschäck, Klöckner, Amthor, Ament & Fengler, 2012). Diese Untersuchungen zeigen eine deutliche Reduzierung des Bahnfehlers bei einer Erhöhung der Taktung der Regelung.

Den eigentlichen Flaschenhals im Basis-System bildet die Verarbeitung im Bereich Ablaufsteuerung und Bahnplanung. Dies begründet sich auf der Verwendung des Realtime-Controllers und den damit verbundenen hohen Kommunikationszeiten. Durch eine Verschiebung der Echtzeitfunktionen dieses Funktionskomplexes auf den FPGA kann die Kommunikationszeit entfallen bzw. deutlich reduziert werden. Dies ermöglicht grundsätzlich die Steigerung der Taktraten und somit eine Verbesserung der applikationsspezifischen Leistungsparameter des Systems.

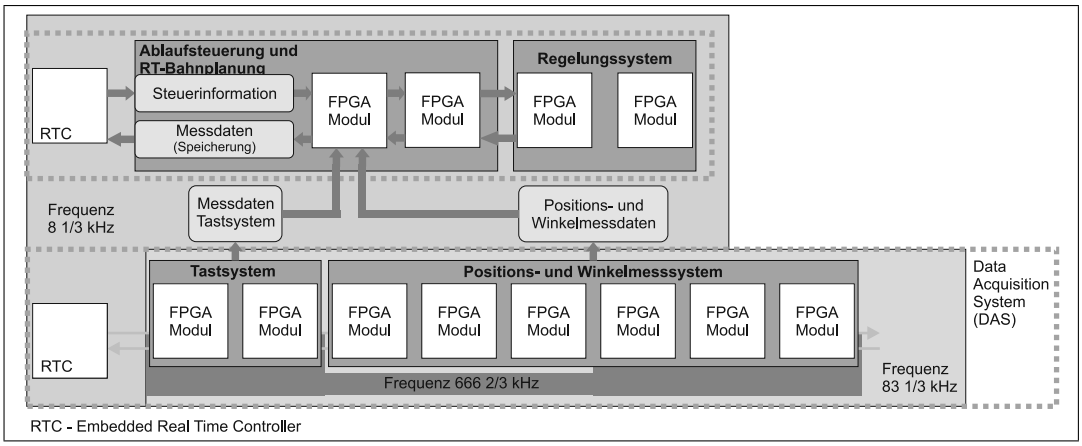


Abbildung 5.3.2: Architektur reduziert auf zwei PXI-Systeme

Eine mögliche Architektur, welche eine rein FPGA-basierte Verarbeitung unterstützt ist in Abbildung 5.3.2 dargestellt. Zunächst erfolgt eine Reduzierung auf zwei Systemkomponenten. Eine Komponente beinhaltet wie bisher die Signalerfassung und Datenvorverarbeitung (Tastsystem, Positions- und Winkelmesssystem) und eine weitere Komponente ist verantwortlich für Ablaufsteuerung, Bahnplanung und Regelung. Die echtzeitrelevante Verarbeitung erfolgt nun ausschließlich innerhalb der FPGA-Module.

Durch die Einsparung eines PXI-Systems erfolgt trotz der Verwendung eines zusätzlichen FPGA-Moduls eine Reduzierung der Hardware-Kosten. Der Verlust eines PXI-Systems ist allerdings auch gleichbedeutend mit einer Reduzierung des gesamten Datenvolumens, welches dem Nutzer zur Verfügung gestellt werden kann. Eine Speicherung von Daten ist in dieser Architektur lediglich auf zwei anstelle von drei RT-Controllern möglich. Ob diese Einschränkung applikationsbezogen relevant ist und Auswirkungen auf das Nutzungsverhalten hat, wird an dieser Stelle nicht bewertet.

5.3.1.3 Architekturvariante mit reduzierten Ein- und Ausgangssignalen

Die Basis-Architektur sowie die Echtzeit-Architektur unterstützen den vollen Umfang an verfügbaren maschinenbezogenen Sensoren und Aktoren. Die Verwendung von einzelnen Sensoren insbesondere im Bereich des Tastsystems sowie der Winkelmessung kann stark vom konkreten Anwendungsfall des Nutzers abhängen. So ist durchaus denkbar, dass Anwender die verfügbaren Schnittstellen nicht vollständig nutzen, da es für die Durchführung ihrer Messaufgaben nicht notwendig ist.

Auf Seiten der Signal- und Datenverarbeitungseinheit resultiert dies in nicht genutzten Ressourcen und ergibt ein Ungleichgewicht beim Verhältnis von Kosten und Nutzen. Eine variable nutzerspezifische Architektur des Signal- und Datenverarbeitungssystems anhand der Anforderungen ist daher wünschenswert. Die unterschiedlichen Anforderungen können auf Variationen im mechanischen Aufbau oder auf Basis von Nutzeranforderungen im Sinne von unterschiedlichen Ausbau- und Leistungsstufen der SDPU beruhen.

Ein Beispiel solch einer reduzierten Architektur ist in Abbildung 5.3.3 dargestellt. Die Einschränkungen bzw. Reduzierungen betreffen das Positions- und Winkelmesssystem sowie das Tastsystem. Bei dem Winkelmesssystem wird lediglich die Winkelmessung über optische Autokollimatoren (AKF) unterstützt. Für die Positionsmessung können maximal vier Interferometerkanäle genutzt werden und zur Anbindung eines Tastsystems stehen lediglich acht analoge Ein- und Ausgangsignale zur Verfügung. Eine weitere Reduzierung wäre auch im Bereich des Regelungssystems denkbar, wird an dieser Stelle allerdings nicht berücksichtigt.

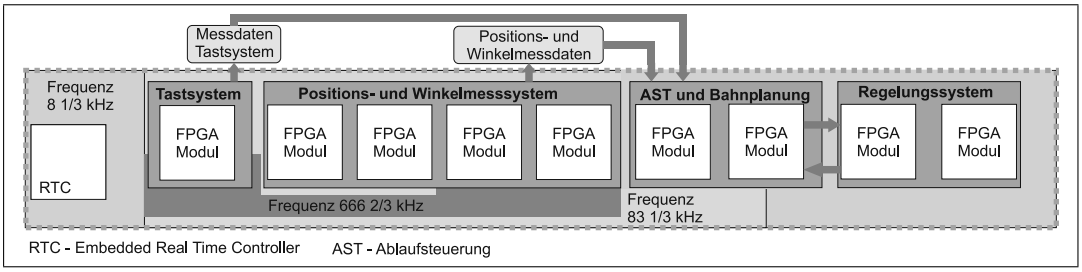


Abbildung 5.3.3: Architektur reduziert auf ein PXI-Systeme

Insbesondere im Vergleich zur Basis-Architektur ist der geringere Hardwareaufwand des Systems offensichtlich. Statt drei PXI-Systemen mit elf FPGA-Modulen kommen lediglich acht FPGA-Module in einem PXI-System zum Einsatz. Hierdurch ergibt sich eine Reduzierung der Kosten für die Signal- und Datenverarbeitungseinheit. Die

interne Verarbeitung unterscheidet sich nur unwesentlich, da die grundlegenden Maschinenfunktionen weiterhin vorhanden sind.

5.3.1.4 Architekturvariante Single System

Neben der Reduzierung der Leistungsfähigkeit und dem damit verbundenen verringerten Hardwareaufwand (Komplexität), können ausgehend von der Variante mit der FPGA-basierten Echtzeitverarbeitung weitere optimierende Schritte durchgeführt werden, die jeweils in unterschiedlichen Systemarchitekturen resultieren. Im Gegensatz zu der Variante mit reduzierten Ein- und Ausgangssignalen, erfolgt keine Reduzierung der Leistungsfähigkeit des Verarbeitungssystems in Bezug auf die unterstützten Funktionen und Schnittstellen. Die Architekturen werden nachfolgend als Single-System-Variante bezeichnet, da sie auf ein PXI-System beschränkt sind. Durch die Vielzahl an FPGA-Modulen bleibt der Charakter eines verteilten Echtzeit-Systems weiterhin erhalten.

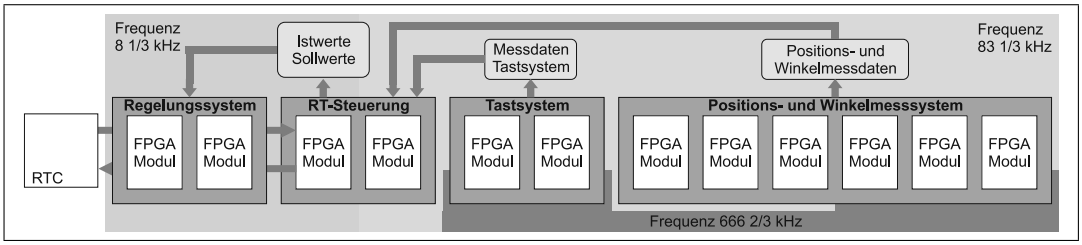


Abbildung 5.3.4: Architektur mit verlustfreier Reduzierung auf ein PXI-System

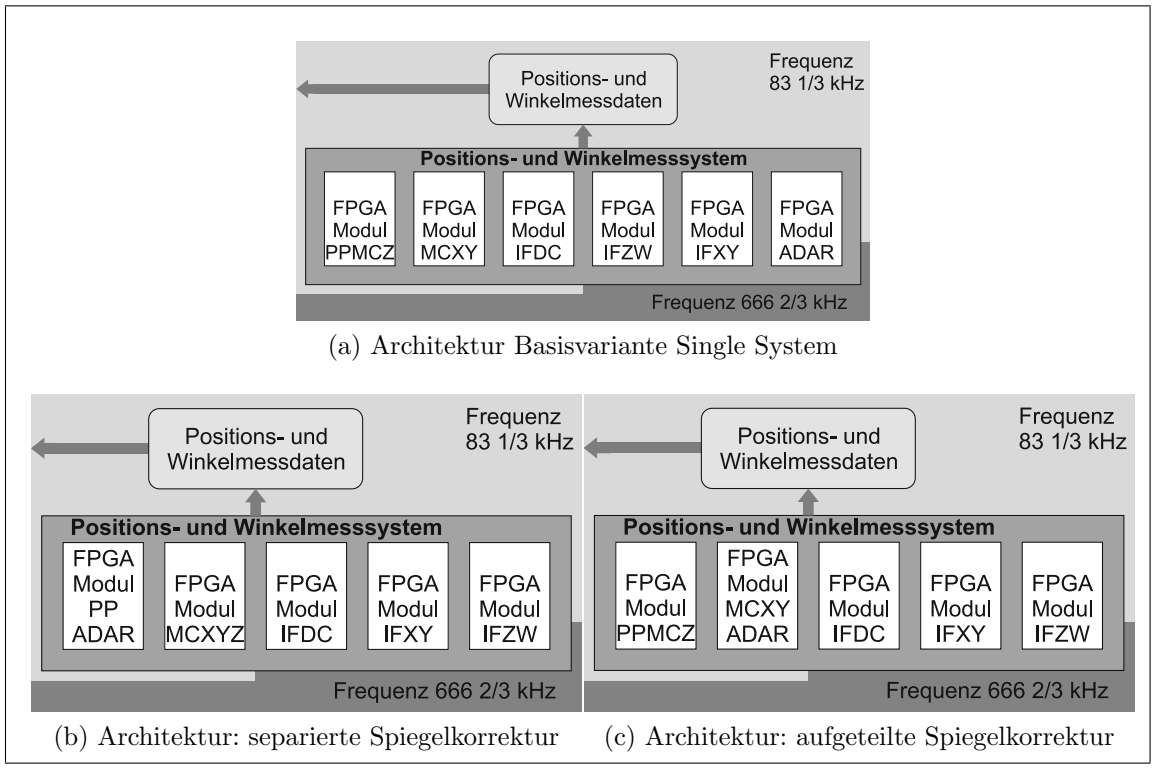
5.3.1.4.1 Architekturvariante Single System (abgeleitete Basisvariante) In Abbildung 5.3.4 ist die Basisstruktur der Single-System-Variante dargestellt. Dies ist eine konsequente Fortführung der Optimierung in Hinblick auf die Reduzierung der Systemkomplexität und führt ausschließlich zu einer Reduzierung der Hardwarekomplexität ohne den Verlust von verarbeitenden Funktionen.

Im Vergleich zu einer Architektur bestehend aus drei PXI-Systemen sind Einschränkungen im Bereich der Datenspeicherung möglich, da für die Erfassung bzw. Speicherung lediglich ein PCI-Bus zur Übertragung von Echtzeitdaten von den FPGA-Modulen verfügbar ist. Auf Simulationsbasis lässt sich dieser Aspekt nicht bewerten, da Details zur Leistungsfähigkeit der Backplane-Kommunikation und Implementierung auf dem RT-Controller fehlen, d.h. die Kommunikation wird von Art und Umfang der auf dem RT-Controller ausgeführten Funktionen (CPU-Last) beeinträchtigt.

Die Anforderungen an die Übertragung von Nutzdaten sind zudem sehr stark abhängig von der konkreten Messaufgabe und den zugehörigen Parametern. Zu diesen Einflussfaktoren zählen das Volumen der zu speichernden Daten sowie die Speicherfrequenz. Die Frequenz variiert in Abhängigkeit zur Messaufgabe, beispielsweise Aufnahme eines einzelnen Wertes, messtakt-basierte Speicherung mit bis zu $83\frac{1}{3}\ kHz$ oder einem Scan, bei dem die Frequenz durch das Speicherraster sowie die Verfahr-Geschwindigkeit bestimmt wird.

Stellt sich im Betrieb des Systems heraus, dass eine Vielzahl von Messaufgaben und Nutzeranforderungen existieren, welche höhere Leistungsparameter im Bereich Speicherung benötigen, kann beispielsweise im einfachsten Fall durch Anbindung von externen Speichersystemen eine Erhöhung der Leistung (Speicherplatz und/oder Datenrate) erfolgen.

5.3.1.4.2 Architekturvariante Single System mit verbesserter Ressourcen Nutzung Ausgehend von der zuvor präsentierten Basis-Variante Single-System gibt es weitere ressourcenbezogene Möglichkeiten zur Optimierung der Systemarchitektur. An dieser Stelle werden stellvertretend zwei solcher Varianten des Teilsystems zur Positionsmessung betrachtet.



(a) Architektur Basisvariante Single System

(b) Architektur: separierte Spiegelkorrektur

(c) Architektur: aufgeteilte Spiegelkorrektur

Abbildung 5.3.5: Architektur des Positionsmessystems

In der Basisvariante 5.3.5a kommen sechs FPGA-Module zum Einsatz, durch eine veränderte Aufteilung der Funktionen innerhalb der FPGA-Module kann wie in 5.3.5b und 5.3.5c dargestellt die Anzahl der FPGA-Module auf fünf für dieses Teilsystem reduziert werden. Die Änderung des Ausführungsortes der Funktionen hat zudem einen Einfluss auf die systeminterne Kommunikation und die Datenpfade. Die Umverteilung der Funktionen führt zu einer besseren Ausnutzung (Auslastung) der FPGA-Ressourcen, dies betrifft insbesondere die verfügbaren analogen Ein- und Ausgangssignale.

Von der Verteilung sind die Funktionen Messwertnachverarbeitung (PP), Korrektur der Messwerte bezüglich der Topologie der Spiegelecke der einzelnen Achsen (MCXYZ) sowie Winkel- und Referenzsystem (ADAR) betroffen. Die FPGA-Module

zur Verarbeitung der Interferormeter (IFXY und IFZW) sowie das FPGA-Modul zur Datenfusionierung (IFDC) ändern sich in den unterschiedlichen Varianten nicht. Zwischen den beiden Varianten gibt es auf konzeptioneller Ebene keine wesentlichen Unterschiede, es existiert lediglich eine leichte Variation der digitalen und analogen Ein- und Ausgänge der einzelnen Module. Ein Vergleich hinsichtlich der relevanten Leistungsparameter erfolgt im Abschnitt 5.3.2.

Als Vorteil der Variante mit einer separierten Spiegelkorrektur, d.h. die Spiegelkorrektur für alle Achsen wird auf einem FPGA-Modul durchgeführt, kann im Hinblick auf eine „user dependend system configuration“ (beispielsweise strukturelle Entfernung der Online-Spiegelkorrektur auf Wunsch des Endanwenders) den Plug&Play-Charakter erhöhen.

5.3.2 Explorative Bewertung der Architekturvarianten

Unter der explorativen Analyse und Bewertung der einzelnen Architekturvarianten wird eine simulationsbasierte Untersuchung auf Basis der erstellten systemspezifischen Verarbeitungs- und Kommunikationsmodelle verstanden. Den Referenzpunkt oder Ausgangspunkt bildet die Architektur des Prototypen der Signal- und Datenverarbeitungseinheit (Abbildung 5.3.1). Eine effektive Untersuchung von möglichen zukünftigen Architekturen ohne aufwändiges Prototyping ist so möglich.

Eine Vielzahl von Variationen der Architektur sind durch die modulare Struktur möglich. Von den im vorigen Abschnitt dargestellten Systemen, werden vier Varianten für den Vergleich ausgewählt:

Single-System-Basic Bei dieser Variante erfolgt die gesamte Verarbeitung innerhalb der FPGA-Module (Abbildung 5.3.4).

MCXYZ Den Ausgangspunkt bildet die Variante Single-System-Basic, die Berechnung der Spiegelkorrektur aller Achsen erfolgt auf einem FPGA-Modul. Ein FPGA-Modul wird durch Umverteilung der einzelnen Funktionen eingespart (Abbildung 5.3.5b).

PPMCZ Das System ist vergleichbar mit der Variante MCXYZ und unterscheidet sich lediglich durch die Verteilung der einzelnen Funktionen (Abbildung 5.3.5c). Die Berechnung der Spiegelkorrektur aller Achsen ist auf zwei FPGA-Module verteilt.

Reduced-IO Dies ist das minimale System, welches sich durch die reduzierte Anzahl an verfügbaren analogen und digitalen Schnittstellen auszeichnet (Abbildung 5.3.3).

Ein zentrales Kriterium für den Vergleich der Systeme ist die erreichbare Zykluszeit, im Prototypen beträgt diese 120 μs (Percle et al., 2011). Die Parameter für die Verarbeitung und Kommunikation, die durch die Datenverarbeitung, Kommunikation und Algorithmen innerhalb der FPGA-Module entstehen, lassen sich aus dem Prototypen entnehmen bzw. in diesem bestimmen. Die einzelnen Modelle werden entsprechend konfiguriert.

Neben diesen Architekturvarianten werden ebenso zwei in den Abschnitten 5.2.2.2.1 und 5.2.2.3 vorgestellte Erweiterungen der Kommunikation berücksichtigt. Dies ist zum einen die Erhöhung des Übertragungstaktes auf 40 MHz (HS-Comm) und zum anderen die Verwendung einer Codierung (Code). Eine Kombination beider Verfahren (Code-HS) wird ebenso betrachtet.

Tabelle 5.3: Vergleich der verschiedenen Architekturvarianten bezüglich der Laufzeit eines Verarbeitungsschrittes von der Messwertaufnahme bis zur Ausgabe der Stellgrößen

Architektur-variante	**standard**	**HS-Comm**	**Code**	**Code-HS**	**Ersparnis**
Single-System-Basic	55.42 μs	46.12 μs	62.66 μs	49.02 μs	ca. −11 T€
MCXYZ	54.75 μs	46.35 μs	61.63 μs	49.10 μs	ca. −17 T€
PPMCZ	54.66 μs	46.26 μs	61.54 μs	49.02 μs	ca. −17 T€
Reduced-IO	56.11 μs	46.40 μs	64.42 μs	49.72 μs	ca. −25 T€
Prototyp	120 μs	–	–	–	0 €

Die Ergebnisse der Simulationen finden sich aufgearbeitet in Tabelle 5.3. Hier zeigt sich deutlich der Vorteil der reinen FPGA-Lösung für die Echtzeitverarbeitung. Die Laufzeiten liegen deutlich unter 60 μs für einen Verarbeitungsschritt von Messwerterfassung bis zur Ausgabe der Stellgrößen. Eine Taktrate von bis zu 20 kHz für das Gesamtsystem ist basierend auf diesen Resultaten möglich. Vergleicht man die einzelnen Varianten untereinander, so zeigt sich der deutliche Einfluss der Kommunikation. Durch eine Steigerung der Übertragungsrate ist eine weitere Verringerung der Systemlaufzeit erzielbar. Die Ergebnisse zeigen auch, dass eine Erhöhung der Übertragungsrate den zeitlichen Mehraufwand ausgleicht, welcher durch die Codierung entsteht. Die erhöhten Anforderungen an die verfügbaren FPGA-Ressourcen werden im Falle der Codierung an dieser Stelle nicht berücksichtigt.

Die einzelnen Architekturvarianten benötigen für die Realisierung eine unterschiedliche Anzahl von Hardware-Komponenten. Im Vergleich zu dem Prototypen verringert sich bei allen vier Varianten die Systemkomplexität, es wird jeweils nur noch ein PXI-System genutzt. Dies ist gleichzeitig mit einer Kostenersparnis verbunden. Als besonders geeignet erscheinen unter den Gesichtspunkten Performance und Kosten die beiden Varianten MCXYZ und PPMCZ. Zur Realisierung einer verbesserten Signal- und Datenverarbeitungseinheit wird die Variante MCXYZ ausgewählt. Auszugsweise werden Komponenten dieser Realisierung in Abschnitt 5.4 vorgestellt.

5.3.3 Darstellung der Modellierung am Beispiel einer Architekturvariante

Im Abschnitt 5.3.2 wurden die Ergebnisse von simulationsbasierten Untersuchungen von zuvor definierten Architekturvarianten vorgestellt. An dieser Stelle werden am Beispiel der Architekturvariante MCXYZ mit einfacher Kommunikation (Variante

standard) auszugsweise die für diese Untersuchungen verwendeten Modelle vorgestellt. Zur Modellierung der Architekturen und Durchführung der Simulationen der unterschiedlichen Szenarien wurde LabVIEW verwendet. Die Modelle der Systemkomponenten (Anwendung und Kommunikation) sowie ein Rahmen für die Simulation mit Beobachterfunktionen wurden erstellt.

Nachfolgend wird zunächst kurz auf die Systemarchitektur eingegangen und die zur näheren Darstellung ausgewählten Komponenten vorgestellt.

Die Systemarchitektur in der Variante „Single System mit separierte Spiegelkorrektur“ besteht aus elf FPGA-Modulen, die miteinander verbunden sind und die vier Teilsysteme Positionsmesssystem (FPGA-Module: IFXY, IFZW, IFDC, MCXYZ, PP_ADAR), Tastsystem (FPGA-Module: IFPS, ADPS), Regelungssystem (FPGA-Module: CXY, CZ) und Echtzeitsteuerungssystem (FPGA-Module: SCDC, TRAJ) realisieren.

5.3.3.1 Ermittlung der Parameter des Systems

Ein wichtiger Aspekt für die Durchführung der Untersuchungen ist die Ermittlung der Kennwerte des Systems zur Einstellung von Modellparametern. Gegenstand der durchgeführten Untersuchungen ist die Betrachtung und Bewertung der unterschiedlichen Systemarchitekturen im Hinblick der Leistungsfähigkeit im Sinne erreichbarer Taktraten sowie Mess- und Regelzyklen. Wichtige Kennzahlen sind somit die Verarbeitungszeiten innerhalb der Anwendungsteile sowie die architekturbedingte Kommunikation.

Für die Verarbeitungszeiten der einzelnen Anwendungskomponenten bestimmen sich diese im Wesentlichen auf Basis der durchgeführten Berechnungen (Algorithmen). Eine Bestimmung der Simulationsparameter ist auf drei verschiedene Weisen möglich: Prototyping der Anwendungskomponente, Messungen am Prototypen der NPMM-200 und Ableitung der Laufzeit aus den Programmen des Softcore-Prozessors.

Die erste Variante umfasst die Erstellung eines Prototypen der Anwendungskomponente und darauffolgend die Bestimmung relevanter Parameter an diesem. Diese erste Variante spielt eine untergeordnete Rolle, da eine vollständige Signal- und Datenverarbeitungseinheit zur Verfügung steht und daher im Allgemeinen ein gesondertes Prototyping der Komponenten nicht erforderlich ist.

Dies leitet direkt zur nächsten Möglichkeit über, die Bestimmung von Kennwerten durch Messungen an dem Prototyp der NPMM-200. Die zugehörige Signal- und Datenverarbeitungseinheit enthält alle Funktionen und Anwendungskomponenten. Daher können die Laufzeiten und weitere Eigenschaften direkt ermittelt werden, soweit diese von außen beobachtet werden können.

Bei der letzten Variante wird eine Vereinfachung vorgenommen. Hierbei wird festgelegt, dass die Verarbeitung einer Anwendungskomponente oder Funktion nur abhängig ist von der Berechnung der Algorithmen. Folglich ist es ausreichend, die Laufzeit der einzelnen Algorithmen zu bestimmen. Die Algorithmen werden innerhalb der Softcore-Prozessoren berechnet. Aus einem zugehörigen Programm kann direkt die Anzahl der Verarbeitungstakte und somit die Laufzeit abgeleitet werden. Diese dritte Variante ist sehr variabel und flexibel, da sie lediglich das Vorhandensein eines Programms für den Softcore-Prozessor voraussetzt. Im Vergleich mit den anderen beiden

Varianten sind an dieser Stelle die Anforderungen am geringsten. Es wird weder eine komplette Signal- und Datenverarbeitungseinheit noch ein funktionsfähiges FPGA-Design erwartet. Gleichzeitig wurde die Leistungsfähigkeit des Softcore-Prozessors im Rahmen umfangreicher Prototyping-Untersuchungen bereits nachgewiesen, so dass für die einzelnen Algorithmen eine Hardware-Implementierung vernachlässigt werden kann.

Die Parameter zur Konfiguration der ausgewählten Architektur sind in Tabelle 5.4 dargestellt. In dieser finden sich in der ersten Spalte die FPGA-Module und in den folgenden Spalten finden sich zum einen die Parameter für die Verarbeitung und die Parameter für die Kommunikation. Die Spalte Softcore ist relevant für die Verarbeitung. Diese gibt an wie viele Prozessortakte für die Berechnung einer Funktion (Ausführung eines Programms) benötigt werden. Dabei können einem FPGA-Modul kein, ein oder mehrere Softcore-Prozessoren zugeordnet sein.

Tabelle 5.4: Parameter für das Simulationsmodell Single-System-MCXYZ

FPGA-Modul	**Empfang**	**Soft-core**	**Senden**	**Paket-anzahl**
IFPS	Clock	810	IFPS.C1 (ADPS.C1)	13
ADPS	Clock ADPS.C1 (IFPS.C1)	310 220	ADPS.C2 (SCDC.C1)	11
IFXY	Clock	810	IFXY.C1 (IFDC.C0)	13
IFZW	Clock	810	IFZW.C1 (IFDC.C1)	13
IFDC	IFDC.C0 (IFXY.C1) IFDC.C1 (IFZW.C1)	-	IFDC.C2 (MCXYZ.C1) IFDC.C3 (SCDC.C0)	14 26
MCXYZ	MCXYZ.C1 (IFDC.C2)	340 240	MCXYZ.C2 (PPADAR.C1)	14
PPADAR	Clock PPADAR.C1 (MCXYZ.C2)	180 950	PPADAR.C2 (SCDC.C2)	22
SCDC	SCDC.C0 (IFDC.C3) SCDC.C1 (ADPS.C2) SCDC.C2 (PPADAR.C2)	-	SCDC.C3 (TRAJ.C1)	60
TRAJ	TRAJ.C1 (SCDC.C3) TRAJ.C2.2 (CXY.C1.2)	750	TRAJ.C2.1 (CXY.C1.1)	32
CXY	CXY.C1.1 (TRAJ.C2.1) CXY.C2.2 (CZ.C1.2)	800	CXY.C1.2 (TRAJ.C2.2) CXY.C2.1 (CZ.C1.1)	54 16
CZ	CZ.C1.1 (CXY.C2.1)	600	CZ.C1.2 (CXY.C2.2)	22

Nach der Bestimmung der Einstellungen für die Verarbeitung fehlen nun die Parameter für die Kommunikation. Ähnlich wie bei der Verarbeitung sind ausführliche Messungen an dem Prototypen der NPMM-200 oder ein Prototyping von geänderten Kommunikationsverbindungen möglich.

Durch die einfachen Mechanismen und die unidirektionale Punkt-zu-Punkt-Verbindungen können die kommunikationsbezogenen Laufzeiten und Verzögerungen von den folgenden Eigenschaften abgeleitet werden: die Anzahl der zu sendenden Daten

(Größe und Datentyp der Datenstruktur), Anzahl der parallelen Datenleitungen und die Taktrate der Clockleitung (Übertragungstakt).

Die Übertragungsrate in der Standard-Variante ist mit 16 MHz spezifiziert. Die weiteren Parameter sind in Tabelle 5.4 dargestellt und werden nachfolgend erläutert. Hier sind für jedes FPGA-Modul die Datenquellen (Spalte Empfang) sowie die Datensenke (Spalte Senden) angeben. Der Aufbau der Architektur und die Verbindung der einzelnen Komponenten sind hierdurch spezifiziert. Bei FPGA-Modulen, welche analoge Signale verarbeiten, ist zusätzlich als Datenquelle Clock angegeben, welche den Basistakt (Zähltakt) repräsentiert.

Jedes FPGA-Modul besitzt zwei oder vier Connectoren mit vierzig digitalen Leitungen, welche für die Kommunikation verwendet werden können und mit C0, C1, C2 oder C3 bezeichnet werden. Des Weiteren können die vierzig Leitungen eines Connectors aufgeteilt werden und zur Realisierung mehrerer Verbindungen genutzt werden. Dies betrifft beispielsweise Connector 2 des FPGA-Moduls TRAJ (TRAJ.C2.1 und TRAJ.C2.1). In der letzten Spalte ist die Anzahl der Kommunikationspakete angegeben, diese ergibt sich aus der Anzahl der Daten und der Anzahl der für die Verbindung genutzten Datenleitungen. Mit der Zusammenstellung der Tabelle 5.4 sind die wichtigsten Parameter für die ausgewählte Systemarchitektur festgelegt.

5.3.3.2 Darstellung des Simulationsmodells

Zunächst ein paar grundlegende Eigenschaften der Modelle. Jedem FPGA ist ein Anwendungsmodell zugeordnet, dieses entspricht in der Simulation einem VI. Als Beispiele für Anwendungsmodelle finden sich nachfolgend in Abschnitt 5.3.3.3 die Modelle für die FPGA-Module ADPS und MCXYZ.

Für jedes FPGA-Modul werden eingehende oder ausgehende Warteschlangen (FIFOs) definiert, welche die Schnittstelle zum Empfangen und Senden von Daten bilden, d.h. dies sind die Schnittstellen zu den Modellen der Kommunikationskomponenten (Communication Controller).

Die Modelle der Kommunikationskomponenten beinhalten dabei die Funktionalität des Communication Controller und des Channels. Jedes Kommunikationselement besitzt eine eingehende (Quelle/Source) und eine ausgehende (Ziel/Destination) Warteschlange. Zur Verbindung zweier FPGA-Module und Aufbau eines Kommunikationskanals werden die Schnittstellen entsprechend zugeordnet. Eine Vernetzung und der Aufbau der Architektur lassen sich so realisieren.

Den Aufbau des Simulationsmodells zeigt das VI in Abbildung 5.3.6. Auf den ersten Blick lassen sich zwei Bereiche erkennen, auf der linken Seite erfolgt zunächst die Konfiguration der Simulation und anschließend erfolgt die Ausführung der einzelnen Modelle. Diese Konfiguration umfasst zum einen das Anlegen der Beobachterfunktion, hier wird eine Datei erstellt, welche die ausgewählten Ereignisse protokolliert, und zum anderen die Erstellung der Warteschlangen (FIFOs) für die Verbindungsendpunkte der FPGA-Module. Diese Endpunkte ermöglichen die Kopplung der einzelnen Anwendungskomponenten und die Kommunikation (Datenaustausch zwischen einzelnen FPGA-Modulen).

Die Ausführung der einzelnen Modelle ist optisch in fünf Bereiche aufgeteilt. Dies

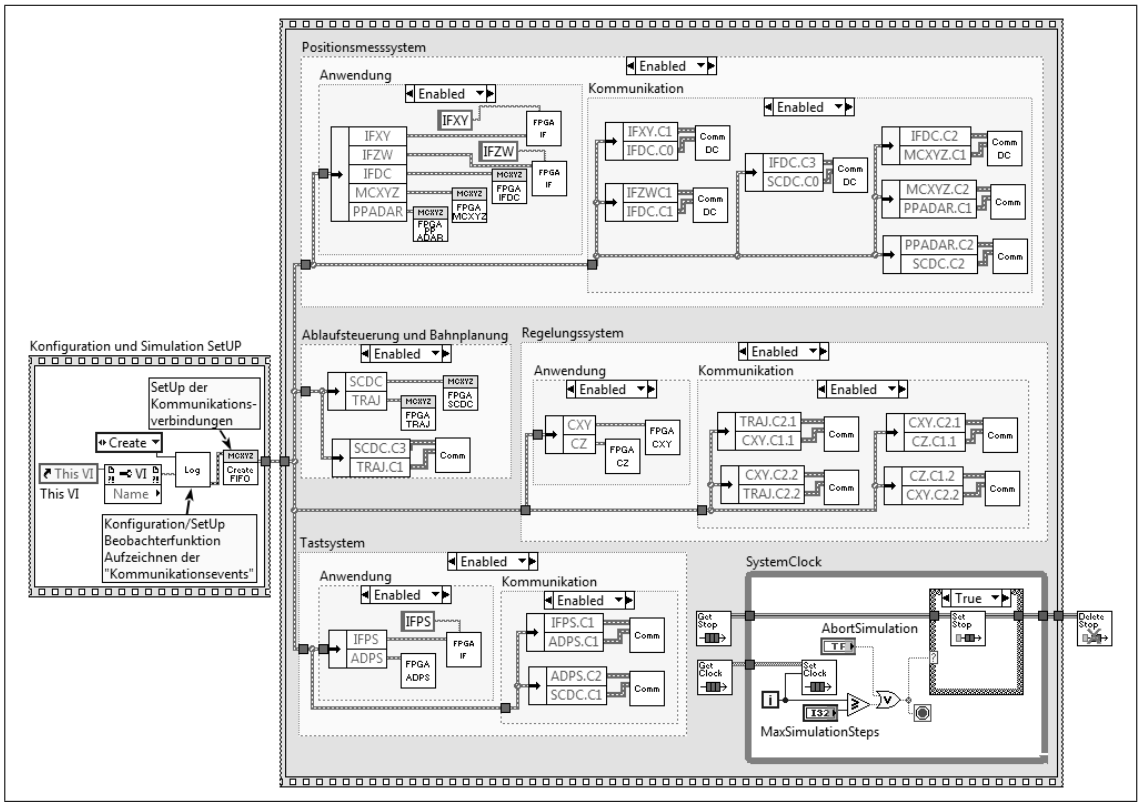


Abbildung 5.3.6: Aufbau des Simulationsmodells für die Systemarchitektur Single-System-MCXYZ

sind zunächst die vier Teilsysteme: Positionsmesssystem, Ablaufsteuerung und Bahnplanung, Regelungssystem und Tastsystem. Der fünfte Bereich dient zur Steuerung der Simulation (Abbrechen der Simulation oder Festlegung einer maximalen Anzahl an Simulationsschritten) und zur Generierung des Signals für den Basistakt zum Einlesen der analogen Signale. Dieses entspricht dem Beginn eines Zähltaktes und triggert die Verarbeitung im System und im Modell.

Die einzelnen Modellbestandteile der Teilsysteme werden am Beispiel des Positionsmesssystems kurz erläutert. Zur Übersichtlichkeit erfolgt auch hier eine Aufteilung in zwei Bereiche, zum einen den Anwendungsteil und zum anderen die Kommunikation. Innerhalb des Anwendungsteils werden die einzelnen Modelle der Anwendungskomponenten (FPGA-Module) des Teilsystems Positionsmesssystem instanziiert und ihnen werden die Warteschlangen zugewiesen. Näheres zum Aufbau einer Anwendungskomponente findet sich nachfolgend am Beispiel der Modelle für die FPGA-Module ADPS und MCXYZ.

In dem Bereich der Kommunikation wird zwischen zwei Verbindungsendpunkten durch die Instanziierung eines Modells des Communication Controller und die Zuordnung von Verbindungsendpunkten ein Kommunikationskanal aufgebaut. Es werden zwei unterschiedliche Modell-Typen für den Kommunikationskanal verwendet: Comm (Communication Standard Variante) und Comm DC (Communication Standard Variante with direct Copy). Der Kommunikationskanal Comm DC modelliert eine direkte Weiterleitung der Daten ohne Beachtung der Anzahl der Pakete die übertragen wer-

den. Näheres zu den Modellen zur Kommunikation findet sich in Abschnitt 5.3.3.4.

Nach der Betrachtung des Top-Level-Models werden nun einzelne ausgewählte Elemente dieser Top-Level-Ebene näher betrachtet.

5.3.3.3 Modelle ausgewählter Anwendungskomponenten (FPGA-Module)

5.3.3.3.1 Modellierung des FPGA-Modul MCXYZ Die Korrektur der sieben Positionswerte (Längenwerte IFX0, IFX1, IFY0, IFY1, IFZ0, IFZ1, IFZ2) bezüglich der Topologie der Spiegelecke erfolgt auf dem FPGA-Modul MCXYZ. Die Topologie der drei Spiegel wird jeweils durch Stützstellen (Raster im Abstand von 1 mm) auf dem FPGA hinterlegt.

Zur Korrektur werden die sieben Längenwerte von dem FPGA-Modul IFDC über die digitale Schnittstelle empfangen. Sobald die Interferometerwerte empfangen wurden wird die Berechnung durchgeführt. Hierzu werden zwei Softcore-Prozessoren eingesetzt (ein Prozessor welcher gleichzeitig die Korrektur für den X- und den Y-Spiegel durchführt und ein Prozessor für die Korrektur des Z-Spiegel).

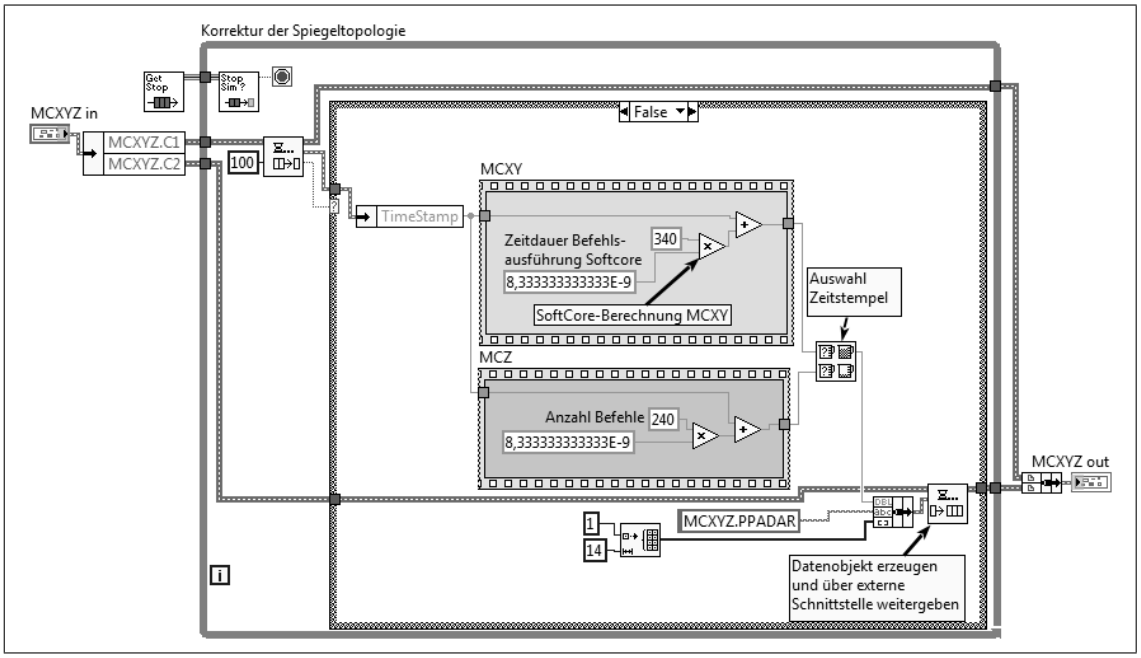


Abbildung 5.3.7: VI des einfachen Ausführungsmodells des FPGA-Moduls MCXYZ

In Abbildung 5.3.7 ist das VI des einfachen Modell des FPGA-Moduls MCXYZ dargestellt. Auf der linken Seite erkennt man die beiden Schnittstellen (FIFOs) für die Kommunikation (MCXYZ.C1 und MCXYZ.2). Dabei dient die Schnittstelle C1 zum Empfangen und die Schnittstelle C2 zum Senden von Daten. Ein Datensatz, welcher über die Schnittstellen ausgetauscht wird, besteht aus einem Zeitstempel, einem Bezeichner und einem Datenarray.

Die Verarbeitung des FPGA-Moduls wird in einer Schleife gekapselt. Sobald Daten empfangen werden beginnt die Verarbeitung. Man erkennt die beiden Softcore-Prozessoren für die Berechnung, zum einen zur Spiegelkorrektur der X- und Y-Achse

(oben, MCXY) und zum anderen zur Spiegelkorrektur der Z-Achse (unten, MCZ). Die Verarbeitungszeit bestimmt sich aus der Anzahl der Befehle der jeweiligen Berechnungen (240, 340) multipliziert mit der Ausführungszeit eines Befehls ($8,3\ ns$, $120\ MHz$). Die resultierenden Zeiten werden mit dem Zeitstempel der empfangenen Daten verrechnet.

Das Ergebnis wird zum Senden weitergeleitet, der Datensatz besteht aus dem Maximum der beiden Zeitstempel (die Daten werden weitergesendet wenn beide Berechnungen abgeschlossen sind), dem Bezeichner für die Kommunikationsverbindung (die Daten werden vom FPGA-Modul MCXYZ zum FPGA-Modul PP_ADAR gesendet - MCXYZ.PPADAR) und einem Array der Größe 14, welches die Nutzdaten repräsentiert.

5.3.3.3.2 Modellierung des FPGA-Modul ADPS Das FPGA-Modul ADPS dient gemeinsam mit dem FPGA-Modul IFPS zur Anbindung von Tastsystemen. Zentrale Aufgabe ist die Erfassung der Signale des Tastsystems. Diese können Interferometer oder andere analoge Signale sein, und ausgehend von den Eingangssignalen unter Verwendung eines Polynoms mit einstellbaren Koeffizienten erfolgt die Berechnung der Auslenkungs- (Deflection), Positions- und Kraftwerte des Tastsystems.

Bei der Verarbeitung wird zwischen Zähltakt ($666\frac{2}{3}\ kHz$) und Messtakt ($83\frac{1}{3}\ kHz$) unterschieden. Die Erfassung der analogen Eingangssignale erfolgt im Zähltakt. Beim Übergang zwischen beiden Takten erfolgt eine einfache Reduktion der Taktrate.

Im Messtakt erfolgt die zweistufige Berechnung. Sobald die acht analogen Werte zur Verfügung stehen erfolgt der erste Teil der Polynomberechnung. Die Längenwerte von bis zu vier Interferometern werden vom FPGA-Modul IFPS über die digitale Kommunikationsschnittstelle gesendet. Ist die Berechnung abgeschlossen und stehen die Längenwerte des FPGA-Moduls IFPS zur Verfügung wird die Berechnung abgeschlossen. Anschließend werden die berechneten Werte zum FPGA-Modul SCDC (Sequence Control Data Collector) übertragen.

Die Ausführung innerhalb des FPGA-Moduls ADPS ist ein wenig umfangreicher als im zuvor betrachteten FPGA-Modul MCXYZ. Die Verarbeitung erfolgt in drei Schritten:

- Die Erfassung der analogen Eingangssignale auf Basis des Zähltaktes.
- Der Übergang zum Messtakt mit Berechnung des ersten Teils der Polynome.
- Der Empfang der Längenwerte und des ersten Teilergebnis mit anschließender Berechnung der Ausgabewerte.

In Abbildung 5.3.8 ist das VI des einfachen Modell des FPGA-Moduls APDS dargestellt. Hier sind zwei Verarbeitungsschleifen zu erkennen. Die obere Schleife repräsentiert die ersten beiden Verarbeitungsschritte und die untere Schleife modelliert den letzten Arbeitsschritt.

Das Taktsignal des Zähltaktes wird im Simulationssystem über eine FIFO verbreitet. Jedes FPGA-Modul inkrementiert bei vorhandenem Taktsignal einen internen Zähler, welcher einen globale gültigen Zeitpunkt repräsentiert. Im linken Teil der

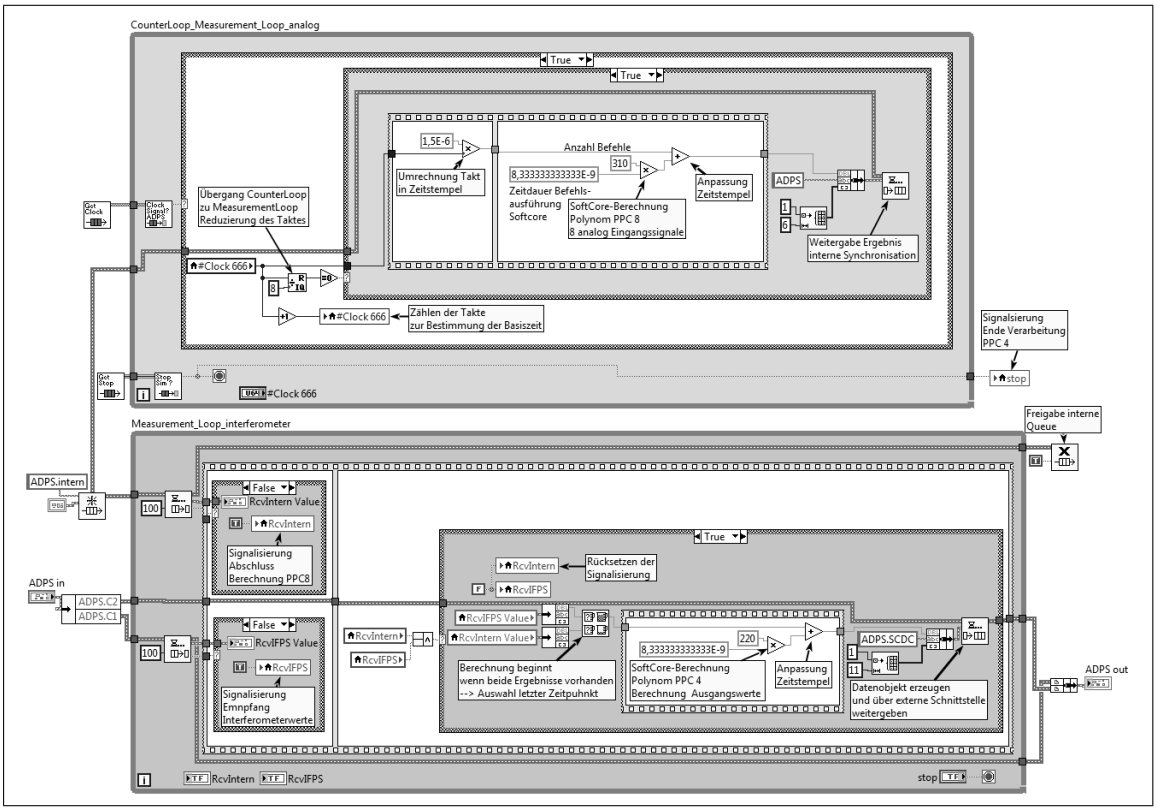


Abbildung 5.3.8: VI des einfachen Ausführungsmodells des FPGA-Moduls ADPS

Verarbeitungsschleife der CounterLoop (oben) wird auf das Auftreten eines Taktsignals gewartet. Der Aktuell Wert des Zählers wird gelesen, geprüft ob ein Messtakt abgeschlossen wurde und der Zähler inkrementiert.

Erfolgt ein Übergang zum Messtakt wird aus dem Taktzähler der aktuelle Zeitstempel bestimmt. Anschließend beginnt der erste Schritt der Berechnung, die Ausführungszeit wird bestimmt und ein Datum mit dem zugehörigen Zeitstempel generiert. Zur Bereitstellung der Daten für den nächsten Berechnungsschritt wird eine interne FIFO verwendet.

In der zweiten Verarbeitungsschleife müssen vor der Durchführung der Verarbeitung auf zwei Datensätze gewartet werden, der Empfang wird mit zwei Schaltern (RcvIntern und RcvIFPS) gekennzeichnet. Erst wenn beide Schalter den Empfang signalisieren beginnt der zweite Schritt der Berechnung mit Bestimmung des zugehörigen Datensatzes mit gültigen Zeitstempel und anschließendem Senden des Datensatzes.

5.3.3.4 Modellierung der Kommunikation

Die Modellierung der Kommunikationskanäle basiert auf der Verknüpfung von Kommunikationsendpunkten, welche durch Warteschlangen (FIFOs) realisiert sind, und der Verzögerung der Daten abhängig von der Größe des Datenpakets. Die Modellkomponente für die Kommunikation in der Standard-Variante ist in Abbildung 5.3.9 dargestellt. Auf der linken Seite des VIs finden sich die drei Konfigurationselemente: Source, zur Festlegung des Verbindungsendpunktes von dem Daten empfangen

werden; Destination, zur Spezifikation des Endpunktes zu dem die Daten gesendet werden; SendFrequency, zur flexiblen Festlegung des Übertragungstaktes. Die Übertragungsrate beträgt in der Standard-Variante 16 MHz und stellt einen Default-Wert dar. Soll eine andere Übertragungsrate verwendet werden, kann dieser Default-Wert überschrieben werden. In der ausgewählten Systemarchitektur wird dieser in dem Top-Level-Modell nicht angepasst.

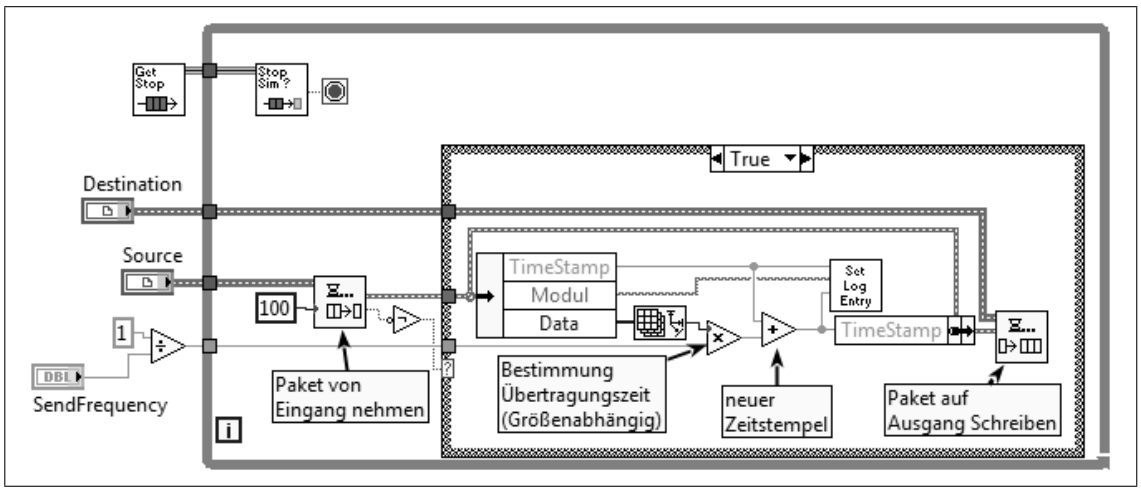


Abbildung 5.3.9: VI des einfachen Kommunikationsmodells

Während der Simulation wird auf ein neues Paket am eingehenden Verbindungsendpunkt gewartet. Wird ein Paket empfangen, wird abhängig von der Übertragungsrate, der Paketgröße und dem aktuellen Zeitstempel des Datenpakets ein neuer Zeitstempel erzeugt. Das resultierende Datenpaket wird zum ausgehenden Verbindungsendpunkt weitergeleitet. Zusätzlich wird das Weiterleiten des Datenpaketes protokolliert.

Für den Vergleich unterschiedlicher Architekturen wurden zusätzlich drei weitere Kommunikationsvarianten (drei Varianten der Protokolle) verwendet: die Highspeed-Variante (HS-Comm), die Variante mit integrierter Codierung (Code) und die Kombination beider Varianten (Code-HS). Die Modelle für diese lassen sich aus der oben dargestellten Umsetzung ableiten. Die Highspeed-Variante unterscheidet sich lediglich durch die Übertragungsrate, diese kann durch einfaches Überschreiben des Default-Wertes ausgewählt werden.

Die Modellierung der Codierung kann mit unterschiedlichen Detaillierungsgraden erfolgen. Der Einfluss der Codierung auf das zeitliche Verhalten entsteht im Wesentlichen durch zwei Aspekte. Dies sind zum einen die zusätzliche Verarbeitungszeit, die für die Durchführung der Codierung und Decodierung notwendig ist, und zum anderen die Vergrößerung des Datenpaketes, durch die Übertragung der zusätzlichen Kontrolldaten. An dieser Stelle wird die in Abschnitt 5.2.2.3.2 vorgestellte Horizontale Codierung ausgewählt, daher kann im ersten Schritt auf die Betrachtung der Verarbeitungszeit verzichtet werden. Somit bleibt als Einflussfaktor die Vergrößerung des Datenpaketes übrig. Die Paketgröße kann durch die Formel $size_{code} = \left\lceil \frac{size}{11} \right\rceil \times 15$ bestimmt werden und wird anschließend zur Bestimmung der Übertragungszeit und Anpassung des Zeitstempels genutzt.

5.3.3.5 Bestimmung der Laufzeit durch Simulation und Beobachterfunktionalität

Nach der Betrachtung des Simulationsmodells und einiger ausgewählter Komponenten verbleibt abschließend die Darstellung der Auswertung der Simulationsdaten und die Bestimmung der Laufzeit eines Verarbeitungsschrittes, wie diese in Tabelle 5.3 aufgeführt werden. Während der Durchführung einer Simulation werden ausgewählte Ereignisse in einer Log-Datei protokolliert. Zu diesen Ereignissen zählen u.a. die Erzeugung von Messwerten, eingehende und ausgehende Daten bei der Kommunikation sowie die Ausgabe der Stellgrößen. Ein Beispiel für die Aufzeichnung dieser Ereignisse zeigt das Listing 5.1.

Listing 5.1: Auszug aus einem Log der Simulation des Systems Single-System-MCXYZ

```
1  0,0000120000   IFXY.C1          Create_MW
2  0,0000120000   IFZW.C1          Create_MW
3  0,0000187500   IFXY.C1          Cin
4  0,0000187500   IFZW.C1          Cin
5  0,0000188750   IFXY.C1          Cout
6  0,0000188750   IFZW.C1          Cout
7  0,0000188750   IFDC.MCXYZ       Cin
8  0,0000188750   IFDC.SCDC        Cin
9  0,0000190000   IFDC.MCXYZ       Cout
10 0,0000190000   IFDC.SCDC        Cout
11 0,0000218333   MCXYZ.PPADAR     Cin
12 0,0000227083   MCXYZ.PPADAR     Cout
13 0,0000306250   PPADAR.SCDC      Cin
14 0,0000320000   PPADAR.SCDC      Cout
15 ...
```

Durch die aufgezeichneten Zeitstempel (in der linken Spalte) und die Kennzeichnung der Ereignisse lassen sich jeweils der Beginn und das Ende eines Verarbeitungsschrittes identifizieren und damit die Ausführungszeit bestimmen. Ein Vergleich der unterschiedlichen Architekturen erfolgt auf Basis der ermittelten Laufzeit.

5.3.4 Abstraktionsebenen

Die zuvor betrachteten Modelle berücksichtigen lediglich das abstrakte Verhalten des Gesamtsystems. Der Abstraktionsgrad befindet sich auf einer sehr hohen Ebene, da sich die Zielstellung der Modellierung und simulationsbasierten Analyse auf grobe Architekturentscheidungen im Hinblick auf die allgemeine Partitionierung des gesamten Systems bezieht. Hierbei bilden die einzelnen Funktionskomponenten, welche innerhalb der einzelnen FPGA-Module realisiert sind, die Basiseinheit bei der Verarbeitung.

Somit erfolgt die Betrachtung von Systemarchitektur und Entwurfsentscheidungen auf der Systemebene. Detailliertere Aspekte werden nicht berücksichtigt, dies sind

beispielsweise Entwurfsentscheidungen im Rahmen der Realisierung und Architektur der Funktionskomponenten und der Realisierung der Kommunikationskomponenten.

Der Vorteil des angewendeten Modellierungskonzeptes ist die einfache Austauschbarkeit von Modellelementen mit unterschiedlichem Grad der Abstraktion, welche zudem die Simulation von Komponentenmodellen mit unterschiedlichen Abstraktionsstufen erlaubt.

Ausgehend von den erstellten Modellen können nun für ausgewählte Komponenten detailliertere Varianten in einzelnen Simulationen Berücksichtigung finden, um den Übergang zu Entwurfsentscheidungen auf Komponentenebene und somit die Berücksichtigung von Implementierungsentscheidungen auf Low-Level-Ebene zu ermöglichen.

Nachfolgend soll das Potential dieser detaillierteren Modellierungsebenen anhand von zwei wesentlichen Aspekten beleuchtet werden: die Funktionskomponenten und die Kommunikation. In diesem Zusammenhang erfolgt die Darstellung von relevanten Fragestellungen, welche durch die Simulation von detaillierteren Modellen lösbar sind.

Der große Vorteil der Simulation gegenüber einem Rapid-Prototyping ergibt sich bei Verwendung von FPGA-Modulen durch verschiedene Aspekte. Zunächst betrifft dies die Erstellung von Prototypen. Die Entwicklungs- und Änderungszyklen auf FPGA-Ebene liegen beim Prototyping bei mehreren Stunden, was sich insbesondere durch die Generierung von Bitfiles ergibt. Dagegen werden auf Modell- und Simulationsebene nur wenige Minuten zur Realisierung veränderter Architekturen oder Variationen von Komponenten benötigt. Ergänzt wird dies zudem durch teure und aufwändige Prototyping-Aufbauten sowie die eingeschränkte und aufwändige Beobachtbarkeit von Signalverläufen.

5.3.4.1 Modellierung von Funktionskomponenten - Ebene der FPGA-Realisierung

Die wesentlichen Aspekte der Realisierung der Funktionskomponenten innerhalb der FPGA-Module sind zum einen die Ausführungszeit und der Ressourcenverbrauch. Die Ermittlung des Ressourcenverbrauchs auf Modellebene ist nur sehr eingeschränkt möglich, dies hängt insbesondere mit der sehr umfangreichen strukturellen Optimierung im Rahmen der Generierung von Bitfiles für den FPGA zusammen. Daher ist die direkte Modellierung und Vorhersage des Ressourcenverbrauchs durch Hinzufügen oder Anpassen von Komponenten aufgrund der Abhängigkeiten von Code-Generierungstools, Optimierungseinstellungen u.ä. nicht möglich.

Die Modellierung und Analyse in Hinblick auf Ressourcenverbrauch wird dementsprechend an dieser Stelle nicht betrachtet. Dies ist durch Einbeziehung der Vorhersage des Ressourcenverbrauchs durch geeignete externe Tools möglich. Beispielsweise ist innerhalb der Xilinx-Tool-Chain sowie den Code-Generierungstools von LabVIEW eine Vorhersage der Ressourcen unabhängig von einer zeitaufwändigen Compilierung des Designs für den FPGA möglich. Die Aussagekraft ist allerdings eingeschränkt, da diese Vorhersage u.U. sehr deutlich von einem resultierenden FPGA-Design nach der Compilierung abweichen kann. Im Folgenden wird sich bei der Betrachtung auf das zeitliche Verhalten beschränkt.

Für das zeitliche Verhalten einer einzelnen Komponente lassen sich die folgenden

relevanten Bestandteile identifizieren. Im Allgemeinen hat eine Komponente in der betrachteten Applikation folgende Aufgaben:

1. Erfassung von Prozessgrößen (analoge und digitale Signale)
2. Erzeugung von Messwerten
3. Empfang von Messwerten anderer Komponenten
4. Messwertverarbeitung
5. Senden von Messwerten

Die Verarbeitung in den einzelnen Schritten erfolgt teilweise in unterschiedlichen Takten, wodurch sich für die einzelnen Bereiche Anforderungen an die Synchronisation und Datenübergabe ergeben.

Von zentraler Bedeutung für das zeitliche Verhalten ist neben dieser Synchronisation der einzelnen Schritte auch die algorithmische Verarbeitung der Messwerte. Für die Realisierung von Teilfunktionen gibt es eine Vielzahl von unterschiedlichen Varianten, insbesondere bei der Umsetzung von Fließkommaoperationen. Diese Operationen können durch unterschiedliche Basis-Komponenten realisiert werden. Zusätzlich gibt es die Möglichkeit der Wiederverwendbarkeit der einzelnen Operatoren. Die Variantenvielfalt der Realisierungen reicht bis zur Verwendung von Softcore-Prozessoren innerhalb der FPGA-Module zur algorithmischen Verarbeitung. Die Dimensionierung von Speichern und Warteschlangen spielt an dieser Stelle eine untergeordnete Rolle.

5.3.4.2 Feingranulare Modellierung der Kommunikation - Einbeziehung der FPGA-Mechanismen

Neben den einzelnen System-Komponenten deren Entwicklung, Optimierung und konkrete Umsetzung durch detaillierte Modelle und Simulationen profitiert, können im Bereich der Realisierung der Kommunikation detaillierte Modelle für die Optimierung eingesetzt werden. Im Bereich der Kommunikation sollen dementsprechend zwei Teilaspekte unter dem Gesichtspunkt der detaillierten Modellierung betrachtet werden. Dazu gehört die detaillierte Modellierung der Umsetzung der Sende- und Empfangskomponenten für die in Abschnitt 5.2.2.2 vorgestellte Kommunikationsschnittstelle sowie die ergänzende detaillierte Modellierung des Datenaustausches zwischen FPGA-Modulen und dem Realtime-Controller über den PXI-Bus (PCI-Bus).

5.3.4.2.1 Kommunikation - Einbindung PCI-Bus In der prototypischen Realisierung der Signal- und Datenverarbeitungseinheit der NPMM-200 erfolgt die Bahnplanung und Echtzeitsteuerung auf dem Realtime-Controller. Da die Signalerfassung, Messwerterzeugung und -vorverarbeitung sowie die Ausgabe von Steuersignalen innerhalb der FPGA-Module stattfindet ist ein echtzeitkritischer Datenaustausch zwischen den FPGA-Modulen und dem RT-Controller notwendig.

Für die Verbesserung der Leistungsfähigkeit der Kommunikation zwischen den Komponenten sind die optimale Größe der Datenpakete sowie die übergeordnete

Steuerung relevant und haben Einfluss auf die Übertragung. Eine weitergehende Analyse ist beispielsweise teilweise mit den in der betreuten Diplomarbeit von Liu (2010) entstandenen Modellen zu den Bussystemen PCI und PCI-Express möglich. Interessant ist die Untersuchung lediglich unter dem Gesichtspunkt der Bewertung eines Hardware Austausches von PCI auf PCI-Express zur Erhöhung der Leistungsfähigkeit der Prototyp-Architektur.

Durch die in Abschnitt 5.3.2 vorgestellten Anpassungen zur Optimierung und Leistungssteigerung ist die Betrachtung der PCI und PCI-Express Kommunikation von geringer Bedeutung und für die Echtzeitverarbeitung nicht mehr von Relevanz.

5.3.4.2.2 Kommunikation - Einbeziehung interner FPGA-Mechanismen Während die detaillierte Betrachtung der Backplane-Kommunikation nur von geringerer Bedeutung für eine optimierte Systemarchitektur ist, so birgt die detaillierte Betrachtung des Datenaustausches zwischen den FPGA-Modulen das Potential für weitere Verbesserungen.

Betrachtet man den zeitlichen Aufwand der einzelnen Datenübertragungen so fallen sie auf der Ebene des Gesamtsystems kaum ins Gewicht. Das Volumen der Verbesserungen, welches durch Optimierungen mit Hilfe von detaillierten Modellen erzielbar ist, ist konsequenterweise beschränkt. Die erzielbaren zeitlichen Ersparnisse liegen vermutlich im Bereich von wenigen Übertragungstakten. Bei einer Einsparung von beispielsweise drei Übertragungstakten bei einer verwendeten Übertragungsrate von 16 MHz entspricht dies weniger als 200 ns. Im Verhältnis zu dem Systemtakt von $8\frac{1}{3}$ kHz mit einer Zykluszeit von 120 μs ist dies natürlich verschwindend gering.

Im Bereich der Teilsysteme, die größtenteils mit einem höheren Takt arbeiten, können auch diese geringen Leistungssteigerungen von Bedeutung sein. Innerhalb des Positionsmesssystem, welches mit einer Taktrate von $83\frac{1}{3}$ kHz arbeitet entsprechen die 200 ns etwa $1,5\%$ der Messtaktdauer von 12 μs. Auch hier natürlich nur eine geringe Ersparnis, welche allerdings u.U. durch Summation und mehrere Kommunikationsschritte über das Einhalten oder das Verletzen von Echtzeitbedingungen entscheiden kann.

In unterschiedlichen Teilbereichen lassen sich Fragestellungen zur Anwendung von detaillierten Modellen zur Optimierung des zeitlichen Verhaltens finden. Innerhalb des FPGA-Moduls existieren unterschiedliche Taktdomänen. Die Synchronisation zwischen diesen einzelnen Bereichen kostet Zeit und kann auf Basis von detaillierten Modellen optimiert werden. Unter diesem Gesichtspunkt kann die FIFO-basierte Datenübertragung zur Kommunikationsschnittstelle betrachtet werden.

Neben dem zeitlichen Aspekt kann im Rahmen der Kommunikation im Zusammenhang mit der FIFO-basierten Synchronisation zwischen der komponenteninternen Verarbeitung und der Übertragung auch der Ressourcenverbrauch optimiert werden. Die Dimensionierung dieser FIFOs kann in Simulationen optimal eingestellt werden, um den Ressourcenverbrauch zu minimieren und Datenverluste durch Überlauf zu verhindern.

Verfeinert man die einzelnen Komponenten noch weiter kommen ab einem Punkt auch Betrachtungen im Sinne des Cyber-Physical-Modelling hinzu, welche zusätzlich die Modellierung der Datenleitung an sich zur detaillierten Untersuchung, Analyse

und Optimierung des Timings und der Auswirkungen von Veränderungen im Rahmen der Datenübertragung notwendig macht. Führt man dies konsequent weiter landet man bei einem virtuellen Prototypen, welcher ggf. sogar den mechanischen Aufbau berücksichtigt.

Bei der Verfeinerung der Modelle muss jeweils zwischen Aufwand und Nutzen der Steigerung des Detailgrades eines Modells abgewogen werden. Denn Voraussetzung insbesondere im Hinblick auf die Leistungsbewertung ist eine aussagekräftige Datenbasis mit Leistungsparametern. Das Verhalten der einzelnen Modellkomponenten muss bekannt sein, entweder auf Basis von Prototyping und Profiling Untersuchungen oder auf Basis einer Performance-Estimation, wie sie in Abschnitt 5.3.2 durchgeführt wurde. Abhängig von den verfügbaren Datenquellen unterliegen die aus den Ergebnissen abgeleiteten Entwurfsentscheidungen gewissen Unsicherheiten.

5.4 Auszugsweise Darstellung der Realisierung der Architekturvariante Single System mit separierter Spiegelkorrektur

Die grundlegende Systemarchitektur der Signal- und Datenverarbeitungseinheit des Prototypen der NPMM-200 wurde in Abschnitt 5.3.1.1 vorgestellt. Ausgehend von der Analyse der ersten Version des Prototypen und aufbauend auf die simulativen Untersuchungen erfolgte die Weiterentwicklung der Signal- und Datenverarbeitungseinheit.

Nachfolgend werden auszugsweise einzelne Komponenten des erweiterten Prototypen, welche in LabVIEW realisiert wurden, präsentiert. Als Systemarchitektur wurde für die Weiterentwicklung die in Abschnitt 5.3.1.4.2 eingeführte Architekturvariante Single System mit separierter Spiegelkorrektur ausgewählt. Nachfolgend wird die Realisierung von drei Systemteilen kurz dargestellt. Eine umfangreiche und ausführliche Darstellung ist an dieser Stelle u.a. aus Gründen der Vertraulichkeit nicht möglich. Die ausgewählten Komponenten sollen die grundlegenden Mechanismen der Verarbeitung verdeutlichen.

Es werden zwei FPGA-Module mit hohen Echtzeitanforderungen betrachtet, zum einen ein FPGA-Modul mit Signalerfassung und Messwertverarbeitung und zum anderen ein FPGA-Modul, welches in erster Linie eine Datenfusion und Datenverteilung vornimmt (Routing). Die dritte Komponente bildet ein Teil der Applikation mit geringeren Anforderungen an die Echtzeit, dieser wird auf dem Realtime-Controller des PXI-Systems ausgeführt.

Die erste Komponente ist in Abbildung 5.4.1 dargestellt. Es handelt sich um das VI für das FPGA-Modul ADPS. Dieses ist ein Teil des Tastsystems und hat zwei wesentliche Aufgaben: die Erfassung von analogen und digitalen Signalen des Tastsystems mit der Ermittlung von Rohdaten und anschließend eine Verarbeitung der Rohdaten mit abschließender Erzeugung der Messwerte.

In der Darstellung sind mehrere abgesetzte Bereiche zu erkennen. Insgesamt sind fünfzehn parallele Schleifen vorhanden (inklusive der Schleifen in den Sub-VIs). Im

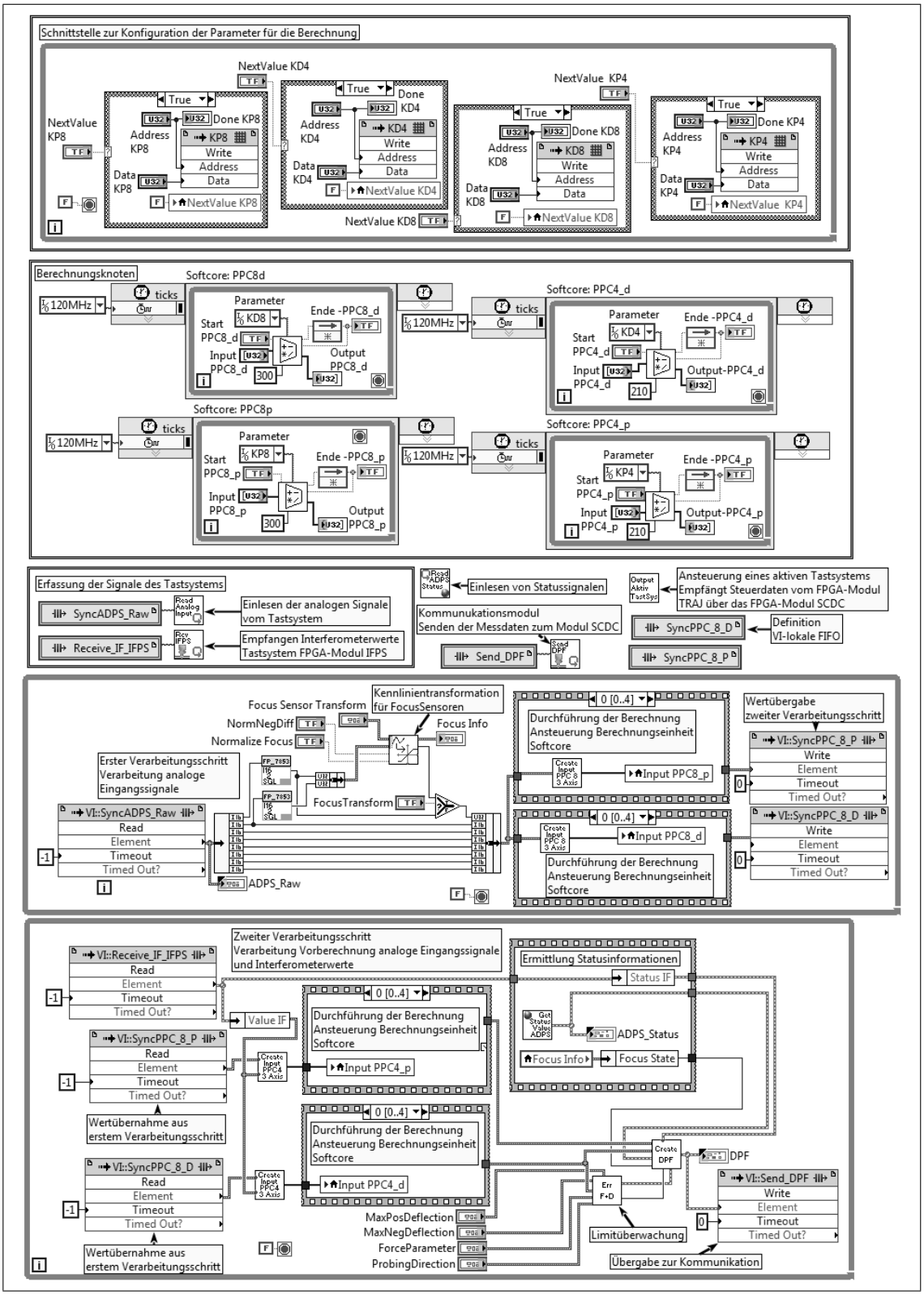


Abbildung 5.4.1: Realisierung des FPGA-Moduls ADPS zur Verarbeitung von Daten eines Tastsystems

oberen Drittel befinden sich zwei Bereiche, welche die vier Berechnungsknoten (zeitgesteuerte Schleifen) zur Durchführung der Messwertverarbeitung sowie vier Schleifen zur Konfiguration und Übermittlung von Parametern an die Verarbeitung beinhalten.

Im mittleren Teil befindet sich die Erfassung der Signale des Tastsystems. Hier erfolgen zum einen das Erfassen der analogen Signale und die Erzeugung der Roh-Messdaten und zum anderen der Empfang der Interferometerwerte vom FPGA-Modul IFPS, welches dem Tastsystem zugeordnete Interferometer auswertet. Der Datenaustausch zwischen den Modulen erfolgt dabei mittels der in Abschnitt 5.2.2.4 vorgestellten Komponenten zur Kommunikation.

Die untere Hälfte enthält die zweigeteilte Steuerung des Ablaufs, welche insbesondere die Ansteuerung der vier einzelnen Berechnungsknoten vornimmt. Vier Berechnungsknoten sind notwendig aufgrund der zweistufigen Verarbeitung sowie aus Gründen der Leistungssteigerung. Zur Verringerung der Berechnungszeit werden in jeder Stufe jeweils zwei Knoten zur parallelen Verarbeitung eingesetzt. Die Berechnungsknoten basieren dabei auf dem in (Pacholik et al., 2011) vorgestellten, selbstentwickelten Softcore.

Im ersten (oberen) Teil der Steuerung erfolgt die Berechnung von Zwischenergebnissen für den zweiten Teil auf Basis der lokal erzeugten Rohdaten. Nach Abschluss des ersten Teils der Berechnung und Empfang der Messwerte vom FPGA-Modul IFPS erfolgen der zweite Berechnungsschritt und die Erzeugung der Messwerte. Vor der Erzeugung der Tastsystem Datenstruktur und der Übertragung zum FPGA-Modul SCDC wird eine Fehler- und Limit-Überwachung durchgeführt.

Die zweite Komponente ist in Abbildung 5.4.2 dargestellt und zeigt das FPGA-Modul SCDC. Zentrale Aufgabe dieser Komponente ist das Empfangen der Messwerte von den einzelnen Messsystemen: Interferometerwerte des Positionsmesssystems (FPGA-Modul IFDC), korrigierte Positions- und Winkelmesswerte des Positionsmesssystems (FPGA-Modul PPADAR) und Messwerte des Tastsystems (FPGA-Modul ADPS). Die drei Kommunikationskomponenten für den Empfang finden sich im oberen linken Teil der Abbildung. Innerhalb eines Verarbeitungsschrittes (untere Schleife innerhalb der Abbildung) können die Messwerte zusätzlich skaliert werden und es können unabhängig von vorherigen Berechnungen neue Geschwindigkeitswerte berechnet werden. Vor der abschließenden Datenfusion und dem Senden der Daten zum FPGA-Modul TRAJ wird eine Überwachung der Geschwindigkeitswerte durchgeführt.

In der Abbildung 5.4.3 ist ein Auszug des Applikationsteils dargestellt, welcher auf dem Realtime-Controller ausgeführt wird. Auf der linken Seite der Abbildung finden sich Komponenten zur Initialisierung und Konfiguration. Zunächst wird die Netzwerkschnittstelle für das Nutzerinterface festgelegt. Danach werden Konfigurationsdaten geladen und anschließend die FPGA-Module initialisiert, konfiguriert und gestartet. Das VI, welches die Referenzen der FPGA-Module erzeugt und Bitfiles den Modulen zuordnet, ist in Abbildung 5.4.4 dargestellt. Hier finden sich alle elf FPGA-Module. Diese werden einzeln geladen und am Ende wird eine Datenstruktur erzeugt, welche die Schnittstelle zum Echtzeitsystem bildet. Eine Übersicht über die Auslastung der einzelnen FPGA-Module kann Tabelle 5.5 entnommen werden, die Angaben stammen aus den Berichten der jeweiligen Generierung der Bitfiles. Die in den FPGA-Modulen

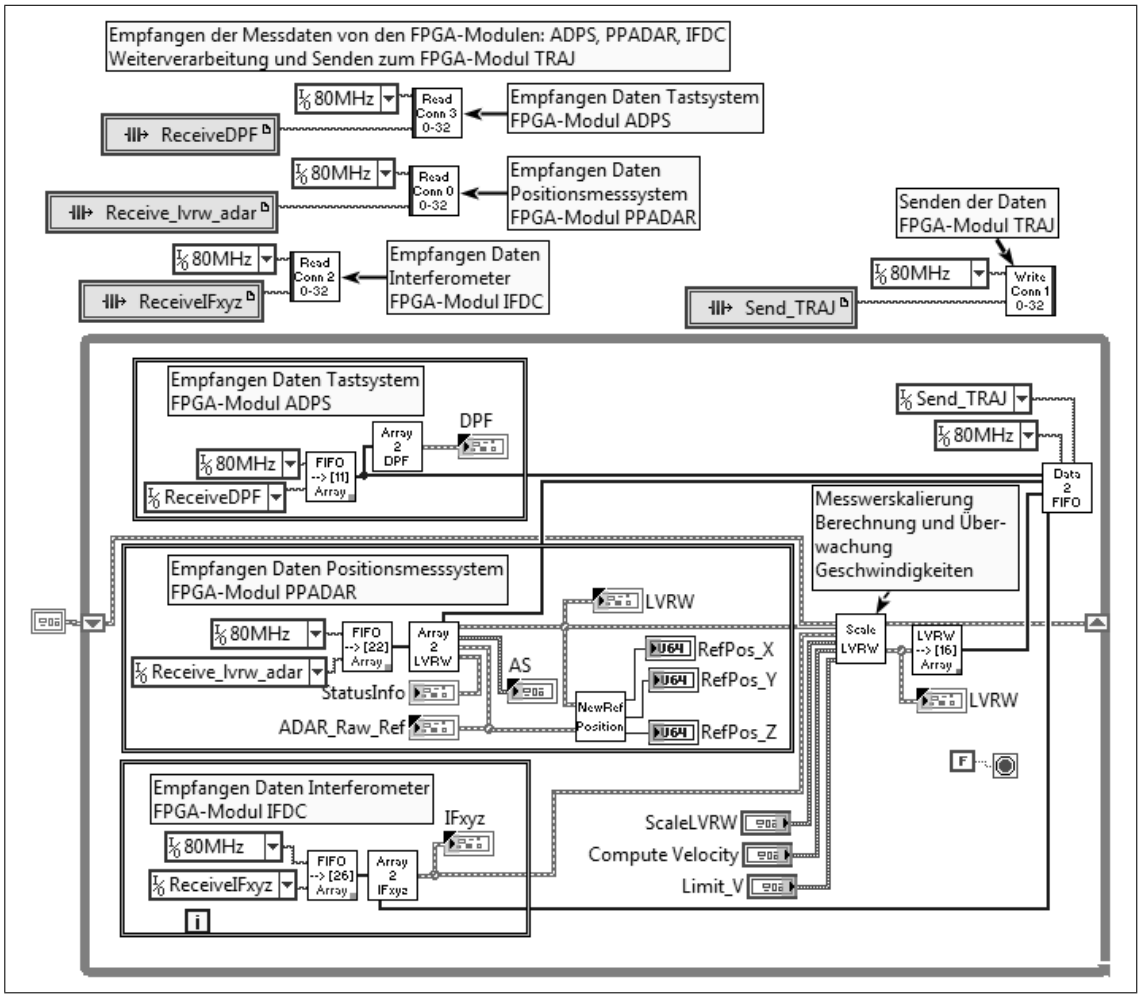


Abbildung 5.4.2: Realisierung des FPGA-Moduls SCDC zur Fusion und Skalierung von Messdaten

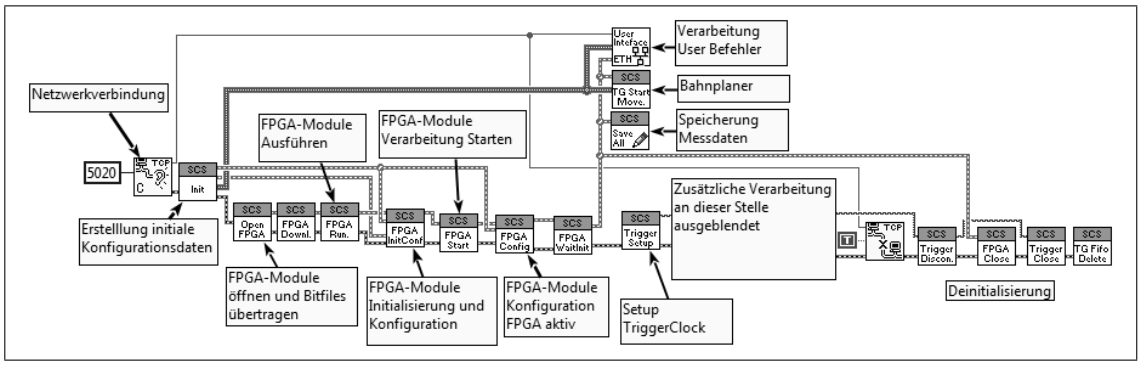


Abbildung 5.4.3: Auszugsweise Darstellung der Realisierung der Applikation auf dem PXI-Realtime-Controller

IFPS, IFXY und IFZW realisierten Algorithmen verwenden trigonometrische Funktionen. Nach der Initialisierung und Konfiguration des Echtzeitsystems gliedert sich die Verarbeitung an dieser Stelle in vier parallele Teile: die Nutzerschnittstelle zur

Verarbeitung von Befehlen über Ethernet, die Trajektorienplanung, die Speicherung von Messdaten und die Ablaufsteuerung mit niedrigeren Anforderungen an die Echtzeit (an dieser Stelle ausgeblendet). Abschließend findet sich auf der rechten Seite die Deinitialisierung des Systems.

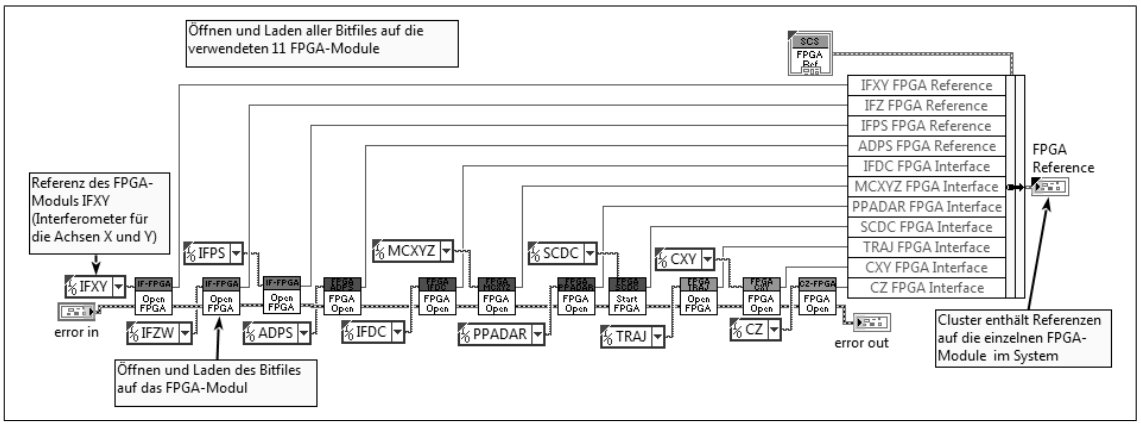


Abbildung 5.4.4: Darstellung des Sub-VI *Open_FPGA* zum Laden der FPGA-Module des PXI-Systems

Tabelle 5.5: Übersicht der verwendeten Ressourcen der einzelnen FPGA-Module

Modul	**FPGA-Typ**	**Datentyp der Operationen**	**Anzahl Softcore Prozessoren**	**Total Slices**	**BlockRAM**
IFPS	PXI-7853R	Single	2	95,4%	25,0%
ADPS	PXI-7853R	Single	4	81,5%	30,2%
IFXY	PXI-7853R	Single	2	95,4%	25,0%
IFZW	PXI-7853R	Single	2	95,4%	25,0%
IFDC	PXI-7813R	-	-	13,1%	4,2%
MCXYZ	PXI-7853R	Double	2	74,3%	81,2%
PPADAR	PXI-7853R	Double	2	91,9%	16,7%
SCDC	PXI-7813R	Double	-	100%	6,2%
TRAJ	PXI-7854R	Double	1	92,5%	18,8%
CXY	PXI-7853R	Double	1	85,4%	32,3%
CZ	PXI-7853R	Double	1	65,8%	35,4%

Der Prototyp der NPMM-200 bildet die Grundlage für die Produktion bzw. Aufbau weiterer Maschinen im Rahmen von Förderprogrammen der DFG (Großgeräteinitiative: „Nanopositionier- und Messmaschinen“)(Deutsche Forschungsgemeinschaft, 2014).

6 Zusammenfassung und Ausblick

Die Arbeit gliederte sich in drei große Bereiche: einen Abschnitt zu den Grundlagen und zwei Hauptteile. Aufbauend auf die Darstellung der Grundlagen des zugrundeliegenden Themenbereichs folgt der erste Hauptteil. Dieser befasste sich mit der Modellierung von Kommunikation, d.h. von Kommunikationsprotokollen und kommunizierenden Systemen. Der zweite Hauptteil setzte sich mit dem Entwurf, der Realisierung und der Weiterentwicklung eines verteilten Systems zur Steuerung einer Nanopositionier- und Nanomessmaschine im obersten Grenzbereich der messtechnischen und informationstechnischen Parameter auseinander. Im Rahmen des Entwicklungsprozesses und der nachfolgenden Analyse und Optimierung des Systems wurden die im ersten Hauptteil vorgestellten Modellierungsmethoden angewendet und dadurch validiert.

6.1 Zusammenfassung

6.1.1 Modellierung von Kommunikationssystemen

Im Rahmen der Grundlagen und des Standes der Technik wurden einführend Grundlagen der bearbeitenden Themenbereiche betrachtet. Die Betrachtung der Kommunikation in der Fahrzeugtechnik umfasste dabei eine allgemeine Klassifikation der Kommunikation in der Fahrzeugtechnik. Hierbei wurde zunächst die SAE-Klassifikation dargestellt und anschließend wurden ausgewählte Kommunikationsprotokolle (LIN, CAN, TTCAN, FlexRay und TTP/C) herausgearbeitet, die eine besondere Relevanz für diese Arbeit haben. Die Relevanz ergab sich dabei zum einen aus der Bedeutung der Domäne Automotive sowie aus der Gegenüberstellung von ereignisorientiertem und zeitorientiertem Buszugriff. Ein wesentlicher Aspekt in verteilten Systemen, insbesondere in Systemen mit einer heterogenen Kommunikationsarchitektur, wie diese in Automobilen vorliegt, bezog sich auf die Kopplung unterschiedlicher Teilsysteme mit Hilfe von Gateways. Die eine Verbindung zwischen unterschiedlichen Teilsystemen herstellen. Im Anschluss wurde die domänenspezifische Kommunikation auf die Automatisierungspyramide abgebildet, welche eine Klassifikation und Strukturierung der Kommunikation im industriellen Umfeld darstellt.

Zwar lag der Schwerpunkt der Arbeit auf der Kommunikation und dem Zugriffsverfahren von unterschiedlichen Bussystemen, aber es ist auch sinnvoll, die Systeme nicht losgelöst von der Applikationsebene zu betrachten. Als ein Teil dieser Applikationsebene kann dabei die Gateway-Funktionalität gesehen werden. Ein weiterer Aspekt, welcher das Verhalten insbesondere das zeitliche Verhalten auf der Applikationsebene beeinflusst, ist das Betriebssystem in einem Steuergerät. Die im Automobilbereich verbreiteten Standards für Betriebssysteme OSEK/OS und OSEKtime/OS wurden

in diesem Zusammenhang kurz aufgeführt.

Abschließend erfolgte die Betrachtung unterschiedlicher Methoden und Ansätze zur Modellierung von Systemen mit dem Schwerpunkt Discrete-Event-Modellierung.

Der erste Hauptteil befasst sich mit der Modellierung von Kommunikationssystemen, im Speziellen mit der Modellierung von Bussystemen aus der Anwendungsdomäne Automotive. Zunächst wurden vier Abstraktionsebenen für die Modellierung der Kommunikationsmodule abgeleitet. Anschließend wurde ein schichten- bzw. komponenten-orientiertes Modellierungskonzept entwickelt, welches sich an den Hardware-Komponenten eines Systems sowie der Komponenten-orientierten Systemdefinition der Domäne orientiert. Innerhalb des Modellierungsansatzes wurden die drei Basiselemente Host, Communication Controller (CC) und Channel definiert. Ausgehend von der grundlegenden Architektur wurde für die Modellierung ein Meta-Modell festgelegt.

Zentraler Ansatzpunkt in dieser Arbeit war, dass für die Analyse des Systemverhaltens spezifische Protokolleigenschaften wie das Zugriffsverfahren eine bedeutende Rolle spielen. Eine abstrakte simulative Systemanalyse, welche beispielsweise ausschließlich die Übertragungsrate berücksichtigt, war nicht ausreichend. Als Beispiel wurden die Bussysteme FlexRay und CAN angeführt und modelliert. Diese stellen aufgrund ihrer charakteristischen Unterschiede durch ihre Ereignis- bzw. Zeitorientierung besondere Herausforderungen im Bereich der Systementwicklung unter Gesichtspunkten wie Teilsystemmigration und Kopplung beider Systemen dar.

Neben der reinen Kommunikation, welche das Hauptaugenmerk auf die Protokolleigenschaften (Element *Communication Controller*) unter Vernachlässigung der physikalischen Schicht (Element *Channel*) legt, wurden Erweiterungen bzw. Strukturierungen auf Ebene der Applikation (Element *Host*) detaillierter betrachtet. Diese bezogen sich zum einen auf die Funktionalität eines Gateways zur Kopplung von Teilsystemen sowie die Verwendung bzw. Berücksichtigung eines Betriebssystems innerhalb der Applikationsebene.

Ausgehend von dem Modellierungskonzept wurden Modelle von Bussystemen betrachtet. Eine Unterscheidung der Modelle erfolgte dabei auf Basis der grundlegenden Abstraktionsebenen. Zunächst wurden Modelle der Abstraktionsebene Level 2 (Modellierung des Buszugriffsverfahrens und interner Protokollabläufe) betrachtet. Für die beiden zeitgesteuerten Kommunikationssysteme TTP/C und FlexRay wurden teilweise Petri-Netz sowie FSM-Modelle definiert und untersucht. Für FlexRay erfolgte auf dieser abstrakten Ebene zusätzlich die Modellierung innerhalb des Multidomänen Tools MLDesigner. Die Einführung von unterschiedlichen Abstraktionsebenen ermöglichte die Simulation von Gesamtsystemen.

Für die weitere Modellierung, welche spezifische Protokolleigenschaften berücksichtigt, wurde ausschließlich das Multidomänen Tool MLDesigner verwendet. Dies begründet sich aus der Unterstützung mehrerer Berechnungsmodelle, die ein großes Potential für Weiterentwicklungen bieten, insbesondere in Hinblick auf die Kombination unterschiedlicher Domänen zur Unterstützung von sehr detaillierten Modellen. Weitere Vorteile sind die einfache Kopplung, Wiederverwendbarkeit, die Berücksichtigung von unterschiedlichen Abstraktionsebenen sowie Komponenten und Hierarchie. Dabei wurden für die Bussysteme CAN und FlexRay mehrere Modelle entwickelt, realisiert und untersucht. Im Falle von FlexRay wurden zudem unterschiedliche Ebenen der

zeitlichen Eigenschaften berücksichtigt.

Die entwickelten Modelle ermöglichten im ersten Schritt lediglich die Untersuchung einzelner Kommunikationscluster. In der Realität finden sich insbesondere in der Domäne Automotive heterogene Kommunikationsstrukturen. Zur Ermöglichung der Untersuchung solcher heterogenen Architekturen wurde eine Modellbibliothek für Gateways (Basis-Architektur für ein Gateway) entwickelt, welche die Kopplung unterschiedlicher Kommunikationscluster erlaubt. Als Referenzbeispiel wurde hier ein FlexRay-CAN-Gateway erstellt.

Mit Bezug auf das Multi-Domänen Tool MLDesigner sind basierend auf dem entwickelten Modellierungsansatz umfangreiche Modellbibliotheken zu den kraftfahrzeugtypischen Bussytemen FlexRay, CAN und LIN entstanden. Der gesamte Ansatz wurde abschließend um eine Modellbibliothek für den Betriebssystemstandard OSEK/OS und OSEKtime/OS erweitert.

Die dargestellten Anwendungsbeispiele zu dem Themenbereich Automotive beschränken sich auf akademische Beispiele. Eine umfangreiche Untersuchung im Bereich der Domäne Automotive war aufgrund der hohen Vertraulichkeit der Systeme innerhalb dieser Domäne nicht möglich. Für die Demonstration der Funktionalität der Modellbibliotheken und deren Anwendung wurde als Systembeispiel die Realisierung eines FlexRay-CAN-Gateway vorgestellt. Zudem wurde die automatische Generierung von Systemmodellen auf Basis der Definition von Kommunikationsclustern mit Hilfe von FIBEX (Standard zur Beschreibung von Netzwerken) demonstriert.

6.1.2 Domänentransfer am Beispiel der modellgestützten Entwicklung der Signal- und Datenverarbeitungseinheit einer Präzisionspositionier und -messmaschine

Innerhalb des zweiten Hauptteils wurden die Erkenntnisse aus dem ersten Teil auf eine weitere Domäne (Mess- und Automatisierungstechnik) erweitert, hier bestand ein stärkerer Bezug zu praktischen Untersuchungen. Für einen reproduzierbaren Prototypen einer Nanopositionier- und Nanomessmaschine mit einem Bewegungsbereich von $200 \times 200 \times 25\ mm^3$ wurde eine Soft- und Hardwareplattform für die Informations- und Signalverarbeitung entwickelt und realisiert. Die Signal und Datenverarbeitungseinheit basierte dabei auf standardisierten Hardware-Komponenten (PXI-Systeme mit mehreren FPGA-Modulen). Bei dem entwickelten System handelte es sich um eine prototypische, reproduzierbare Lösung als Vorläufer für eine Kleinserie von drei Maschinen.

Die Leistungsparameter der Signal- und Datenverarbeitungseinheit korrelieren mit den Qualitätseigenschaften der gesamten Maschine. Das Ziel ist eine möglichst geringe Tast- und Regelzeit des Systems. Die angestrebten Leistungsparameter können dabei nur über eine verteilte Architektur erreicht werden. Für den Datenaustausch zwischen den einzelnen Komponenten gelten hohe Anforderungen an die Echtzeitkommunikation. Eine Modellierung insbesondere zur Bewertung der Leistungsfähigkeit des Systems ist daher notwendig.

In einem ersten Schritt wurden die Anforderungen an die Signal- und Datenverarbeitungseinheit ermittelt und ein grundlegender Entwicklungsprozess mit einem

domänenbezogenen modellbasierten Ansatz betrachtet. Im Rahmen des dargestellten Ansatzes konnte das entwickelte Modellierungskonzept insbesondere im Rahmen der Architekturbewertung und auch zur Vorhersage der Leistungsfähigkeit von Architekturvarianten verwendet werden.

Durch die große Anzahl der zu verarbeitenden analogen und digitalen Signale in Verbindung mit einer vergleichsweise hohen Tastfrequenz und der verteilten Verarbeitung (verteilte Steuerung und hohe Vermaschung von Teilfunktionen) ergaben sich besondere Anforderungen an den Datenaustausch und somit auch die Kommunikation im System.

Nach der Bewertung einiger bestehender Kommunikationsprotokolle erfolgte die Modellierung und Entwicklung von Kommunikationskomponenten, welche die Anforderungen an die Kommunikation innerhalb der Signal- und Datenverarbeitungseinheit erfüllen. Dabei wurden mehrere Protokolle modellbasiert entworfen, bewertet und implementiert. Insbesondere wurden auch die Sicherheit (Safety) der Kommunikation und die Integration von Kodierungsverfahren in die Kommunikationsprotokolle betrachtet und realisiert. Diese konnten abhängig von konkreten Anforderungen flexibel in die Kommunikation integriert werden. Aufbauend auf dieser Entwicklung von Komponenten für die Kommunikation erfolgte die Modellierung, Entwicklung und Analyse von Systemarchitekturen der Verarbeitungseinheit. Die Verwendung des entwickelten Modellierungsansatzes wurde dabei am Beispiel von Entwurfsentscheidungen innerhalb des Teilsystems zur Regelung demonstriert.

Das Resultat dieser ersten Projektphase ist ein die Anforderungen erfüllender Prototyp der Signal- und Datenverarbeitungseinheit für die Nanopositionier- und Nanomessmaschine 200. Die entwickelte Systemarchitektur mit ihren Leistungsmerkmalen wurde 2012 mit dem Graphical System Design Award in der Kategorie Advanced Control Systems von National Instruments gewürdigt (Balzer et al., 2012).

In einer folgenden Projektphase wurden die Analyse des Prototypen in Verbindung mit Optimierungsmöglichkeiten und Strukturanpassungen betrachtet. Zu diesem Zweck wurden unterschiedliche Architektur-Varianten definiert. Im Sinne einer Performance-Estimation erfolgte eine simulationsbasierte explorative Bewertung der Architekturvarianten unter Verwendung der entwickelten Modelle. Berücksichtigt wurden dabei Variationen in der Verteilung von Funktionen, verschiedene Ausprägungen der Systemarchitektur und Variationen im Bereich der Kommunikation.

Die Leistungsfähigkeit und Vorteile der simulativen, modellbasierten Bewertung von Architekturvarianten wurde unter dem Gesichtspunkt unterschiedlicher Abstraktionsebenen dargestellt.

Auf Basis des Prototypen und der Ergebnisse der modellgestützten Systembewertungen erfolgte eine Weiterentwicklung der Signal- und Datenverarbeitungseinheit. Abschließend wurden ausgewählte Komponenten dieses realisierten Systems vorgestellt. Gleichzeitig erfolgte mit dieser Arbeit der erfolgreiche proof of concept der entwickelten Methoden.

Die Leistungsfähigkeit und Relevanz von Nanopositionier- und Nanomessmaschinen mit dem Arbeitsbereich von $200 \times 200 \times 25\ mm^3$ werden im Rahmen einer Großgeräteinitiative „Nanopositionier- und Messmaschinen“ der Deutschen Forschungsgemeinschaft gewürdigt (Deutsche Forschungsgemeinschaft, 2014). Das Gesamtergebnis ist erst durch die entwickelte leistungsfähige Architektur und Kommunikationsstruktur

möglich.

6.2 Ausblick

Im Rahmen dieser Arbeit sind aufbauend auf das entwickelte Modellierungskonzept eine Vielzahl von Modellen für Kommunikationskomponenten entstanden. Die primäre Domäne zur Anwendung oder Verwendung der Modelle ist das Segment Automotive. Die Nutzung des Konzeptes in anderen Domänen wurde an einem mehrstufigen Entwicklungsprojekt aus dem Bereich Prozessdatenverarbeitung und Maschinensteuerung demonstriert. Ausgehend von den erzielten Ergebnissen können weitergehende offene Frage- und Aufgabenstellungen identifiziert werden. Diese werden nachfolgend in mehrere Themenbereiche aufgeteilt.

Der erste Themenbereich kann als „Evaluierung der Domäne Automotive“ bezeichnet werden. Eine Vielzahl der entwickelten Modelle befasst sich mit System-Komponenten welche im Bereich Automotive eingesetzt werden. Hierzu zählen insbesondere die Bussysteme FlexRay und CAN. Die Neuentwicklung des Bussystems FlexRay bildet dabei den initialen Motivationsanstoß für die modell- und simulationsgestützte Systemanalyse in Fällen der Einführung von neuen Technologien. Unabhängig von teuren Prototypen sollen die Auswirkungen der Einführung von neuen Technologien im Speziellen im Bereich der Kommunikationskomponenten auf bestehende und zukünftige Systemarchitekturen betrachtet werden. Im Falle von FlexRay ist insbesondere der Semantikwechsel von ereignisbasierter Kommunikation (CAN) zu einer zeitbasierten Kommunikation (FlexRay) interessant.

Die Verwendung und Anwendung der Modelle im Rahmen von konkreten Projekten und Fragestellungen in der anvisierten Anwendungsdomäne Automotive und damit verbunden die Evaluierung und Bewertung der Modellbibliotheken in einem praxisnahen Umfeld ist eine der zentralen zukünftigen Aufgabenstellungen. In diesem Zusammenhang sollte auch eine ausführliche Betrachtung von AUTOSAR sowie der Ausbau der automatischen Generierung von Modellen erfolgen.

Die Fragestellungen der Technologie Migration welche in dieser Arbeit mit den beiden Kommunikationsbussen CAN und FlexRay thematisiert wurde, ist eine stetig präsente und aktuelle Herausforderung in der Systementwicklung. Dies wird durch neue Bussysteme wie beispielsweise die Weiterentwicklung des CAN mit dem CAN FD (CAN with flexible Data Rate) deutlich.

Die Realisierung von Modellen für weitere Kommunikationsbusse (CAN FD) bzw. Kommunikationskomponenten erweitert die Nutzbarkeit der bestehenden Bibliotheken. Auszugsweise seien hier stellvertretend der Bereich Entertainment und die fahrzeugübergreifende Kommunikation unter dem Schlagwort Car2Car genannt.

Nach der Modellierung im Bereich Automotive soll das ausführlich betrachtete Beispiel einer Signal- und Datenverarbeitungseinheit einer Präzisionsmessmaschine als Ausgangspunkt für zwei weitere Richtungen der Weiterentwicklung dienen. Zunächst betrifft dies den Themenbereich Performance-Estimation von zukünftigen Systemarchitekturen oder von Protokollen in frühen Phasen der Entwicklung. Dieser Aspekt wurde innerhalb der Darstellung des Entwicklungsprozesses kurz aufgenommen und betrachtet. Eine systematische und umfassende Betrachtung der Performance-Esti-

mation inklusive Bewertungsmetriken sollte sich den durchgeführten grundlegenden Untersuchungen anschließen. An dem konkreten Applikationsbeispiel können die entsprechenden Konzepte, Verfahren und Methoden evaluiert werden.

Zudem lässt sich das entwickelte System im weitesten Sinne in den Themenbereich Automatisierungstechnik und Industriebusse einordnen. Die Anwendbarkeit des Modellierungsansatzes konnte an dem konkreten Beispiel demonstriert werden. Durch die Neuinterpretierung von Systemen und die Neuordnung der Kommunikation, die in der Automatisierungstechnik mit dem Zukunftsprojekt „Industrie 4.0“ (BMBF, 2012) verbunden ist, lässt sich die Herausforderung des Transfers und der systematischen Untersuchung des Modellierungsansatzes in Hinblick auf die Anwendungsdomäne Automatisierungstechnik in Verbindung mit „Industrie 4.0“ ableiten.

Ein Beispiel für einen systematischen Entwurfsprozess wurde im Rahmen der Entwicklung der Signal- und Datenverarbeitungseinheit vorgestellt. Hier leistet die Arbeit einen Beitrag in den Bereichen Prototyping und Profiling sowie Performance-Estimation. Die Modelle sollen dabei einen Beitrag zur prototyp-unabhängigen Gewinnung von Performance Informationen durch eine simulations- und modellbasierte Performance-Estimation leisten. Eine große Herausforderung besteht hier in der Automatisierung dieses Entwicklungsschrittes. Von zentraler Bedeutung ist die Verknüpfung von Prototyping und Profiling, Performance-Estimation und dem Management der ermittelten Leistungsparameter (Performance Information). Anhand der Daten erfolgt die modellbasierte Bewertung von Systemarchitekturen. Bei den einzelnen Leistungsparametern muss zudem die Qualität und Aussagekraft oder Verlässlichkeit berücksichtigt werden, aus der sich danach auch die Qualität oder Güte der Leistungsfähigkeit der Systemarchitektur ableiten lassen.

Anhang

A.1 FlexRay Datenframe

Tabelle A.1: Aufbau FlexRay Datenframe (vgl. FlexRay Consortium, 2005b)

Bestandteil	Länge	Verwendung
Header-Segment		
Reserved-Bit	1 Bit	unbenutzt
Payload-Preamble-Indicator	1 Bit	gesetzt: Frame enthält NMV (Statisches Segment) Frame enthält Message ID (dynamisches Segment)
Null-Frame-Indicator	1 Bit	gesetzt: Frame enthält gültige Nutzdaten im Payload-Segment
Sync-Frame-Indicator	1 Bit	gesetzt: Frame wird für die Uhrensynchronisation verwendet
Start-Frame-Indicator	1 Bit	gesetzt: Frame ist Startup-Frame
Frame-ID	11 Bit	Wert (1-2047) entspricht dem Zeitschlitz des Kommunikationszyklus, in dem das Frame gesendet wird
Payload-Length	7 Bit	Wert (0-127) entspricht der Länge des Payload-Segments in 2 Byte Schritten
Header-CRC	11 Bit	Enthält CRC-Prüfsumme von Teilen des Header-Segments
Cycle-Count	6 Bit	Wert (0-63) entspricht dem aktuellen Kommunikationszyklus
Payload-Segment		
Data 0 bis Data 254	0-254 Byte	gerade Anzahl Bytes an Nutzdaten, entspricht dem doppelten des Payload-Length-Werts des Headers
Network Management Vector (optional)	0-12 Byte	nur Statisches Segment, die ersten bis zu 12 Byte eines Statischen Frames enthalten NMV-Daten statt Nutzdaten, wenn PPI im Header gesetzt
Message ID (optional)	2 Byte	nur Dynamisches Segment, wenn PPI gesetzt
Trailer-Segment		
Trailer-CRC	24 Bit	Prüfsumme über das ganze Frame

A.2 Weitere Bussysteme

A.2.1 LIN - Local Interconnect Network

A.2.1.1 Allgemein

Der LIN-Bus ist ein Kommunikationssystem für die serielle Datenübertragung im unteren Kosten- und Leistungsbereich und findet vor allem Verwendung im so genannten Komfortbereich, der sich durch niedrige Übertragungsraten und einen hohen Kostendruck auszeichnet. Entworfen wurde der LIN-Bus als Subbus für den CAN-Bus. Im Jahr 1999 wurde die erste Spezifikation veröffentlicht und liegt aktuell in der Version 2.2 vor. Die Darstellung des Bussystems basiert auf der Spezifikation (LIN Consortium, 2010) und (Grzemba & Wense, 2005; Zimmermann & Schmidgall, 2011)

Das Kommunikationssystem befindet sich in einem evolutionären Prozess, das heißt es wird immer weiter entwickelt. Jede Weiterentwicklung, Änderung oder Erweiterung geschieht unter dem Gesichtspunkt der früheren Zielsetzung, d.h. der Charakter des Protokolls bleibt erhalten.

Die LIN-Bus Spezifikation sieht die Realisierung der Schichten 1, 2, 4 und 7 des ISO/OSI-Referenzmodells vor.

A.2.1.1.1 Busstruktur Beim LIN-Bus, die Struktur eines LIN-Clusters ist in Abbildung A.2.1 dargestellt, handelt es sich um einen logischen und physikalischen Bus bestehend aus einem Master und mehreren Slaves. Innerhalb des Systems wird zusätzlich zwischen Master-Task und Slave-Task unterschieden. Ein Master beinhaltet Master- und Slave-Task, während der Slave lediglich einen Slave-Task enthält. Der Master-Task steuert durch das Senden von Botschaften-Headern die Kommunikation. Der Slave-Task ist verantwortlich für das Senden von Datenbotschaften, als Reaktion auf einen Botschaften-Header. Das Senden von Datenbotschaften erfolgt somit getrennt von dem Senden der Botschaften-Header.

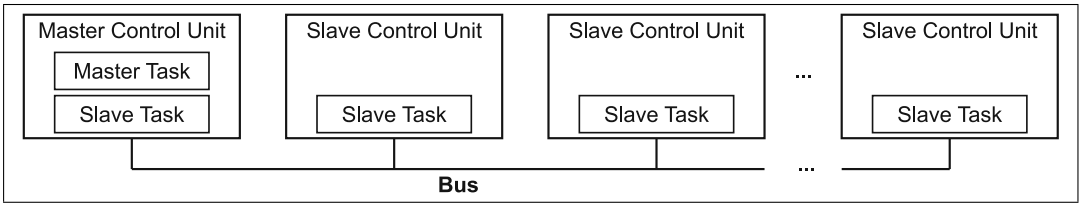


Abbildung A.2.1: Struktur eines LIN-Clusters (vgl. LIN Consortium, 2010)

Für die elektrische Übertragung der Daten wird eine Eindrahtleitung verwendet. Hierbei sind Übertragungsraten zwischen $1 kBit/s$ und 20 $kBit/s$ möglich (in Europa üblicherweise in $9,6$ und $19,2$ $kBit/s$).

Innerhalb des Protokolls gibt es nur grundlegende Mechanismen zur Fehlererkennung (Bit-Fehler, Prüfsummenfehler, Paritätsfehler im Identifier, keine Slave-Antwort, Synchronisationsfehler, keine Bus-Aktivität). Die Fehlerbehandlung selbst erfolgt auf Anwendungsebene und ist größtenteils dem Master zugeordnet. Die einzelnen Slaves

speichern erkannte Fehler, diese werden aber nicht direkt dem Master gemeldet. Diese Informationen können allerdings vom Master angefordert werden. Es erfolgt keine Bestätigung von korrekt empfangenen Nachrichten.

A.2.1.2 Buszugriffsverfahren

Die Kommunikation erfolgt nach dem Client-Server-Modell und wird durch den Master initiiert. Eine Nachricht besteht aus einem vom Master-Task erzeugten Header und einer Response des Slave-Task. Drei Kommunikationsbeziehungen werden unterschieden: ein Slave sendet eine Antwort zum Master, der Master sendet eine Nachricht und ein Slave sendet eine Antwort zu einem anderen Slave.

Der Ablauf der Kommunikation ist innerhalb des Masters durch einen tabellarischen Nachrichten-Schedule hinterlegt. Die Scheduling-Tabelle gibt Reihenfolge und Periode der Nachrichten vor. Die Nachrichten werden über einen 6-Bit Identifier im Header identifiziert.

A.2.1.2.1 Nachrichten-Typen (Frame-Typen) Das Protokoll unterscheidet sechs unterschiedliche Frame Typen. Drei davon haben eine gesonderte Funktionalität (Diagnostic Frames, User-defined Frames und Reserved Frames). Die anderen drei Typen dienen zur Signalübertragung und unterscheiden sich durch das Request-Response Verhalten. Unconditional Frames (Standard Frames) dienen zur einfachen Signalübertragung, bei der auf einen Header genau eine Response eines Slave-Tasks erfolgt. Im Gegensatz dazu dienen Event-Triggered-Frames (Ereignisgesteuerte Frames) zur kombinierten Abfrage von mehreren Ereignissen. Hier können auf einen Header mehrere Responses von Slave-Tasks erfolgen. Eine Antwort wird nur gesendet, wenn ein neues Ereignis vorliegt. Zur Unterscheidung enthält das Datenfeld einen zusätzlichen Identifier zur Nachrichtenkennzeichnung. Dadurch sind die Signale auf maximal sieben Daten-Byte beschränkt. Durch Mehrfachantworten sind Kollisionen möglich, welche vom Master erkannt werden und durch eine Einzelanforderung der Signale behoben werden. Der letzte Nachrichtentyp sind Sporadic Frames (Spontane oder Sporadische Frames), diese dienen als Platzhalter und eine konkrete Nachricht wird innerhalb des Masters ausgewählt.

A.2.1.2.2 Nachrichten-Aufbau Eine Nachricht, wie in Abbildung A.2.2 zu sehen, setzt sich aus einem Message Header (Master-Task) und einer Response (Slave-Task) zusammen. Der Header besteht aus einer Synchronisation Break, einem Synchronisation Field und einem Identifier Field. Die Response besteht aus 1-8 Data Fields und einem Checksum Field.

Die Übertragung der einzelnen Felder basiert auf byte-orientierten LIN-Rahmen. Diese sind UART-8N1 codiert und bestehen aus einem dominanten Startbit, einem Datenbyte und einem rezessiven Stoppbit. Einzelne Rahmen sind durch eine Pause voneinander getrennt (Interbyte Space).

Nachrichten-Header Das Synchronisation Break kennzeichnet den Beginn einer neuen Nachricht durch eine Synchronisationslücke. Anschließend wird, durch das Sen-

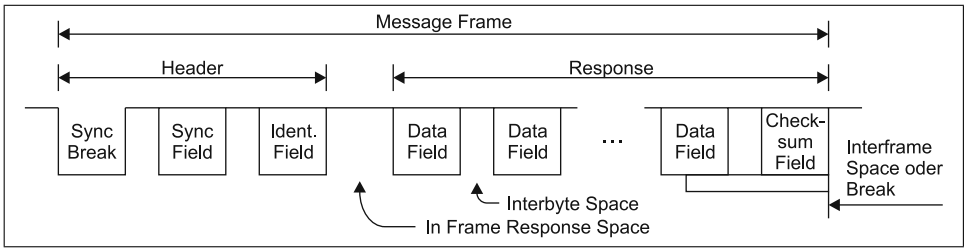


Abbildung A.2.2: Aufbau einer LIN-Nachricht (Message Frame) (vgl. Zimmermann & Schmidgall, 2011)

den eines spezifischen Bitmusters (0x55) innerhalb des Synchronisation Field, die Synchronisation der Teilnehmer (Bittiming) durchgeführt.

Der letzte Teil des Headers bildet das Identifier Field. Dieses beinhaltet den 6-Bit Nachrichten Identifier zur Festlegung von Inhalt und Länge der Nachricht. Die Anzahl der Datenbyte ist abhängig von der Konfigurationsdatei (LIN Descritpion File). Die übrigen beiden Bits (Bit 6 und Bit 7) werden als Parity Bits verwendet („Protected Identifier“).

Getrennt werden Header und Response durch den In-Frame Response Space, dies ist eine Pause mit variabler Länge. Der Sendebeginn der Daten ist veränderlich die Echtzeitqualität der Daten ist dadurch beschränkt.

Nachrichten-Response Die Antwort durch einen Slave-Task setzt sich zusammen aus einem Data Field und einem Checksum Field. Das Data Field ist der erste Teil der Slaveantwort und besteht aus 1-8 LIN-Rahmen. Die Anzahl der Datenbyte ist implizit durch den Identifier bestimmt oder im LDF beschrieben. Die Übertragung beginnt jeweils mit dem LSB (least significant Bit). Die Bestimmung der Checksumme erfolgt durch eine invertierte Modulo-256-Summe über die Datenbytes oder üder die Datenbytes und den Identifier.

A.2.1.3 Übergeordnete Verarbeitungsschichten

A.2.1.3.1 Schicht 4 - Transport Layer Der Transport Layer ermöglicht eine Datenübertragung von Daten mit bis zu 4095 Byte. Die Schicht bietet in erster Linie Dienste zur Konfiguration von Knoten sowie zur Identifikation und zur Diagnose. Seit der Protokollversion 2.0 ist es möglich so genannte „Off-the-Shelf-Knoten“ zu verwenden. Dieses sind Standardknoten, die erst im Zielsystem konfiguriert werden. Die Kommunikation basiert auf einer PDU-Struktur (Protocol Data Unit) und wird durch Services realisiert. Es werden unterschiedliche Typen von PDUs unterschieden.

A.2.1.3.2 Schicht 7 - Application (Host) Layer (Anwendungsdienste) Es gibt drei unterschiedliche Typen von Anwendungsdiensten. Die signal-basierte LIN Core API unterscheidet sechs Dienstgruppen (Driver and Cluster Management, Signal Interaction, Notification, Schedule Management, Interface Management, User Provided Call-Outs) und bietet Dienste und Funktionen zur Initialisierung, Verarbeitung und

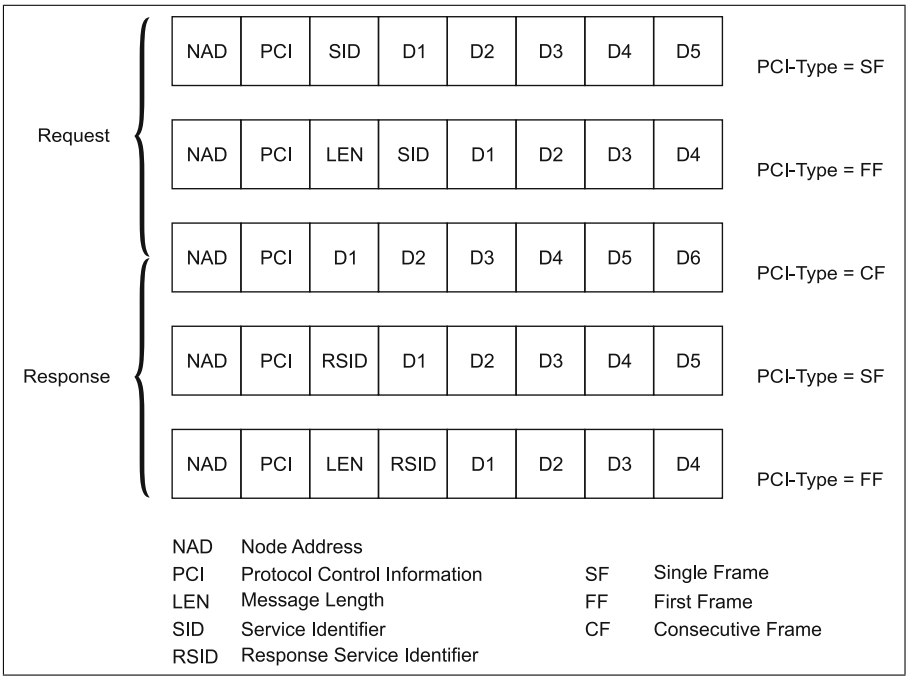


Abbildung A.2.3: Aufbau einer PDU - Transport Layer (LIN Consortium, 2010)

Interaktion mit dem LIN-Kern (Protokoll). Die service-basierte LIN node configuration and identification API unterscheidet zwei Dienstgruppen (Node Configuration, Identification) und basiert auf einem Request-Response-Mechanismus, der über die Transportschicht realisiert wird. Die LIN transport layer API (Diagnose API) erlaubt eine nachrichten-basierte Kommunikation und realisiert eine Schnittstelle für den Austausch von Diagnoseinformationen.

A.2.2 TTP/C - Time-Triggered Protocol Class C

Der Ursprung des TTP (Time Triggered Protocol) liegt an der Universität Berlin. Hier ging es 1979 aus einem Projekt hervor (MARS - Maintable Architecture for Real Time Systems). TTP ist ein zeitgesteuertes Kommunikationsprotokoll für verteilte und fehlertolerante Echtzeitsysteme. Die Entwicklung wurde an der Universität Wien auch in Zusammenarbeit mit einigen Industriepartnern fortgeführt. Mit TTP/C wurde 1993 ein Protokoll mit einem Uhrensynchronisations- und einem Membership-Service veröffentlicht, welches die Anforderungen in den Bereichen Sicherheit, Verfügbarkeit, Zuverlässigkeit, Fehlertoleranz und hohen Datenübertragungsraten erfüllt. Die Markteinführung erfolgte fünf Jahre später. Die Weiterentwicklung erfolgte im Wesentlichen durch die Firma TTTech. TTP/C ist der direkte Konkurrent von FlexRay und es finden sich zwischen beiden Protokollen auch einige Parallelen. (Kopetz & Bauer, 2003) Die Darstellung des Protokolls beruht auf der Spezifikation (ttaGroup, 2002) sowie weiterführender Literatur (Kopetz, 2001; Obermaisser, 2012; Kopetz & Bauer, 2003; Poledna et al., 2001; Lisner & Kessler, 2001).

A.2.2.1 Busstruktur - Time-Triggered-Architecture

In der Time-Triggered-Architecture setzt sich ein Knoten aus drei Komponenten zusammen: einem Host, einem Communication Controller (CC) und einem Input/Output-Subsystem. Die Schnittstelle zwischen Host und CC bildet das Communication-Network-Interface (CNI). Die Schnittstelle zwischen Input/Output-System (Entgegennahme der Sensor- und Aktorsignale) und dem Host bildet das Controlled-Object-Interface (COI). Es existieren zwei Kommunikationskanäle, die Knoten zu einem Kommunikationscluster verbinden. Dieser kann als Bus-, Stern- oder als Mischtopologie ausgeführt sein. Als physikalisches Medium kommen verdrillte Zweidrahtleitung oder Lichtwellenleiter zum Einsatz und ermöglichen eine Übertragungsrate von bis zu 25 $Mbit/s$.

A.2.2.2 Buszugriffsverfahren

A.2.2.2.1 Kommunikationszyklus Der Zugriff auf das Busmedium erfolgt anhand eines TDMA-Verfahrens. Hierfür ist eine globale Zeitbasis der Knoten erforderlich. Die oberste Ebene bildet eine TDMA-Runde (Zyklus) bestehend aus Slots, welche über Macroticks definiert werden. Ein einzelner Macrotick ist in allen Knoten eines Kommunikationsclusters gleich groß. TDMA-Runde, Slot und Macrotick sind globale Einheiten. Die lokale Zeiteinheit Microticks bestimmt einen Macrotick und wird aus einem knotenlokalen Taktgeber (Oszillator) gewonnen. Die Konfiguration und Steuerung des clusterweit konsistenten Schedule erfolgt über die jeweils knotenlokale Message Descriptor List. Diese enthält lokale und clusterweite Konfigurations- und Laufzeitparameter. Die Sendezeit wird periodisch in Slots eingeteilt, welche Knoten zum Senden von Nachrichten zugewiesen werden.

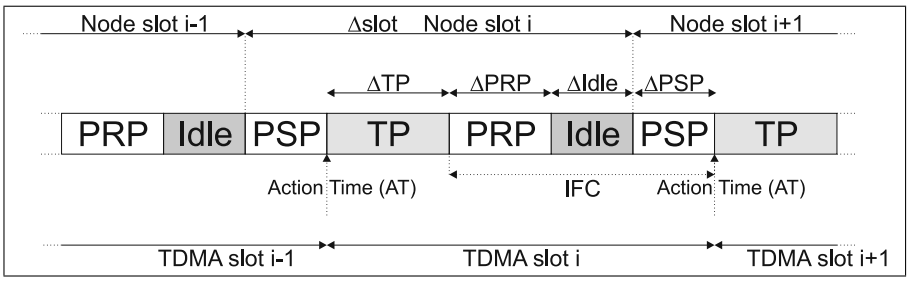


Abbildung A.2.4: TTP/C Kommunikationszyklus (vgl. ttaGroup, 2002, S.33)

Ein Slot besteht aus vier Abschnitten: Pre-Send Phase (PSP), Transmission Phase (TP), Post Receive Phase (PRP) und Idle. Die PSP bereitet das Senden und Empfangen innerhalb des Slots vor. In der anschließenden Transmission Phase überträgt der dem Slot zugeordnete Teilnehmer seine Nachricht, welche von den anderen Teilnehmern empfangen wird. Die Verarbeitung und Auswertung der empfangenen Daten erfolgt in der PRP. Da die Länge von PSP, TP und PRP variieren kann, wird die Idle Phase zur Anpassung des Zeitschlitzes an die vorgegebene Länge verwendet.

A.2.2.2.2 Frame Format Innerhalb des TTP/C werden zwei grundsätzliche Frame-Typen unterschieden, Frames die den Controllerstatus enthalten (z.B. Global Time, Cluster Mode), hierzu zählen X-Frame, I-Frame und Cold Start Frame. N-Frames dagegen beinhalten den Status lediglich in der CRC.

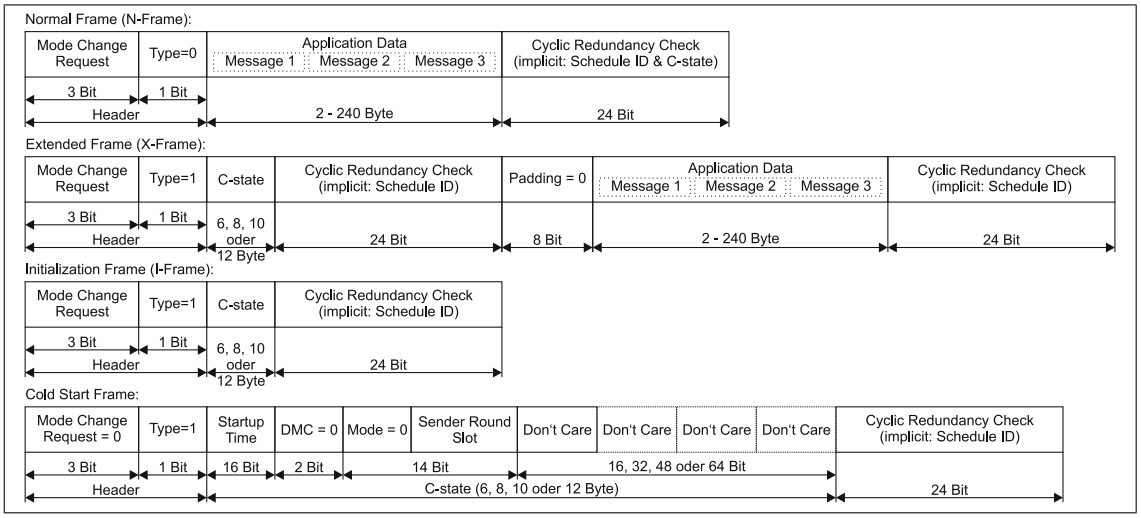


Abbildung A.2.5: TTP/C Frame (Jentzsch, 2003)

Die I-Frames werden bei der Cluster-Initialisierung und der Wiedereingliederung von Knoten genutzt, für diesen Zweck enthält der I-Frame Informationen zu Moduswechsel, Frametyp, Betriebsmodus, Zyklusposition und Membership-Status. X-Frame und Cold Start Frame werden im Rahmen der Clusterinitialisierung eingesetzt. Die Übertragung einfacher Nutzdaten erfolgt mit N-Frames. Der N-Frame besitzt einen Header mit Modus- und Typinformationen, diesem schließen sich die Nutzdaten mit einer flexiblen Datenlänge von 2-240 Byte an. Den Frame-Abschluss bildet die 24 Bit Prüfsumme.

A.2.2.2.3 Cluster-Startup Für den Start eines Clusters und die Herstellung einer synchronisierten Zeitbasis werden mindestens zwei Knoten benötigt. Zu Beginn initialisiert sich der Knoten, anschließend beobachtet der Knoten den Bus, um sich bei Vorliegen einer bestehenden Kommunikation (Empfang von Frames mit Controllerstatus oder Cold Start Frames) in den Ablauf zu integrieren. Falls kein Busverkehr vorliegt und der Knoten den Cluster starten darf, sendet der Knoten in dem ihm zugeordneten Slot den Cold Start Frame.

A.2.2.2.4 Uhren Synchronisation Die gemeinsame synchrone Zeit ist die Basis für das TDMA-Verfahren. Da die lokalen Zeitgeber (Quarzoszillatoren) driften, ist eine Synchronisation der Uhren notwendig. Innerhalb des Protokolls erfolgt die Uhrenkorrektur auf Grundlage der Differenz von tatsächlichen und erwarteten Frame-Ankunftszeiten nach dem Fault-Tolerant Average Algorithmus. Welche Slots für die Zeitberechnung verwendet werden wird in der Message Descriptor List festgelegt.

A.3 Ergänzungen zu den Standards OSEK-OS und OSEKtime-OS

A.3.1 OSEK-OS: Ausgewählte Dienste und Funktionen

Innerhalb von OSEK-OS werden u.a. folgende Dienste und Funktionen zur Verfügung gestellt: Events, Alarme, Messages, Ressourcen, Interrupts. Events werden von Extended Tasks unterstützt und werden von Tasks sowie der ISR erzeugt. Sie dienen zur Synchronisation, zum Taskwechsel und zur Kommunikation zwischen Tasks und ISR. Tasks können im Gegensatz zu ISR auf eine Anzahl von Events warten.

Beim Ablauf eines Alarms stehen Dienste zur Taskaktivierung, zum Setzen von Events und Aufruf von Callback-Funktionen zur Verfügung. Zusätzlich stehen Funktionen zum Beenden oder der Statusabfrage von Alarmen bereit. Es steht mindestens ein Counter zur Verfügung, dieser wird von einem Timer abgeleitet.

Das Ressourcen-Management koordiniert und verwaltet die Taskzugriffe. Es wird sichergestellt, dass zwei Tasks nicht auf eine Ressource zugreifen, sich Prioritäten nicht umkehren und keine Deadlocks auftreten. Für diese Verwaltung ist das OSEK Priority Ceiling Protokoll von zentraler Bedeutung. Beim Ressourcenzugriff können Interrupts gesperrt werden. Die Abarbeitung einer mehrfachbelegten Ressource erfolgt nach dem LIFO-Prinzip (Last In First Out). Der Scheduler stellt für alle Tasks eine Ressource dar, welche Interrupts empfängt und verarbeitet.

Die Anzahl der Interrupts ist abhängig von der verwendeten Hardware. Es werden zwei Kategorien unterschieden. Die erste Gruppe von ISR benutzen keine Betriebssystemfunktionen (ausgenommen Interrupt-Sperrung und -Freigabe). Sie haben keinen Einfluss auf die Taskverwaltung und erzeugen einen geringen Overhead. Die zweite Kategorie bekommt vom Betriebssystem eine Laufzeitumgebung in Form eines ISR-Rahmens. In diesen Rahmen können beliebige Anwender-Funktionen integriert werden, welche einem Interrupt zugeordnet werden.

A.3.2 OSEKtime-OS: Ausgewählte Dienste und Funktionen

A.3.2.1 Dienste und Funktionen

Zur Einbettung anwenderspezifischer Funktionen stehen ISR-Frames zur Verfügung, deren Einrichtung erfolgt bei der Konfiguration des Betriebssystems. Zur Festlegung muss ein Zeitintervall des Auftretens definierbar sein. Innerhalb der Frames können Systemfunktionen verwendet werden. Das Sperren und Freigeben der Interrupts erfolgt durch das Betriebssystem.

A.3.2.2 Kommunikation

Die OSEK-FTCOM Spezifikation beinhaltet eine minimale Funktionalität für die Inter-Task-Kommunikation. Innerhalb des OSEKtime-Systems ist die fehlertolerante Kommunikation ein wichtiger Bestandteil, diese unterstützt eine fehlertolerante Kommunikationsschicht, Echtzeitprotokolle und -systeme. Angeboten werden standardisierte Schnittstellen für Dienste und Funktionen zur Kommunikation. Hierzu zäh-

len globale Nachrichtenbehandlung, Start-up und Reintegrationsunterstützung und einen Dienst zur Synchronisation mit externen Zeitgebern. Zudem lassen sich zwei Subsysteme identifizieren: das fehlertolerante Subsystem und das Subsystem für die Kommunikation zwischen verteilten Komponenten.

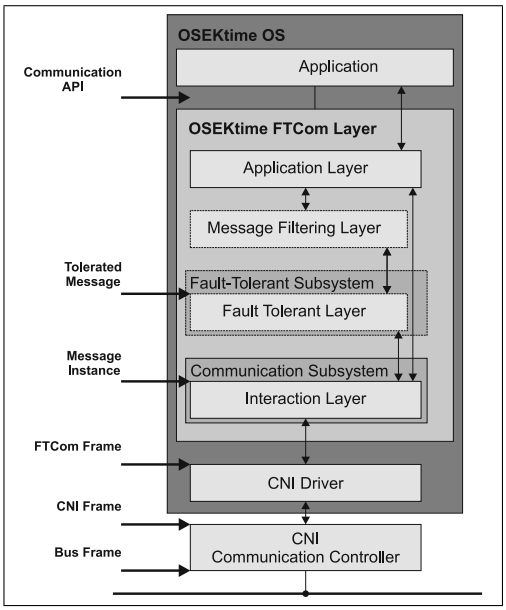


Abbildung A.3.1: Kommunikationssystem OSEKtime-FTCOM (OSEK/VDX, 2001a)

Der Schichtenaufbau des Kommunikationssystems ist in Abbildung A.3.1 dargestellt. Der Application Layer bildet die Schnittstelle zur Anwendung. Dieser untergeordnet ist der Message Filtering Layer und nachfolgend der Fault Tolerant Layer, hier werden Dienste für die fehlertolerante Funktionalität bereitgestellt, z.B. Entscheidungsmechanismen zur Nachrichtenverwaltung, Unterstützung von Statusinformationen. Abschließend bildet der Interaction Layer und der CNI-Driver (Communication Network Interface) die Schnittstelle zum Netzwerk, hier werden beispielsweise die Nachrichten für die verschiedenen Sender bzw. Empfänger angepasst und Dienste für den Datentransfer zur Verfügung gestellt.

Die Nachrichtenschnittstelle innerhalb der FTCOM-Schicht funktioniert ähnlich wie der Transfer von „unqueued"-Nachrichten. Es gibt einen internen und einen externen Kommunikationsweg. Der gewählte Kommunikationsweg bleibt der Anwendung verborgen. Der Datenaustausch zwischen den Tasks ist ausschließlich über die FTCOM-API möglich

A.4 Modelle mit spezifischen Protokolleigenschaften: Local Interconnect Network (LIN)

Neben den in Abschnitt 4.1.3 vorgestellten Modellen für die Bussysteme CAN und FlexRay existiert zudem ein Modell für das Master-Slave-Protokoll LIN. Die Kommunikation zwischen den einzelnen Teilnehmern in diesem Modell erfolgt auf Basis

von Frames. Dieses abstrakte Modell des LIN lässt sich der Ebene *Level 2* zuordnen, welches protokoll-interne Funktionen berücksichtigt. Zusätzlich wird ähnlich wie bei dem Modell zum CAN-Bus ein abstraktes Fehlermodell unterstützt. Realisiert wird das Modell im Multi-Domänen-Tool MLDesigner unter Verwendung der Domänen DE und FSM. Die internen Protokollabläufe werden ebenso wie im CAN-Modell mit Hilfe der Zustandsmaschinen (FSM) modelliert (vgl. Abbildung A.4.3). Die Umsetzung des Modells erfolgte im Rahmen der betreuten Studienarbeit von (Zhu, 2009).

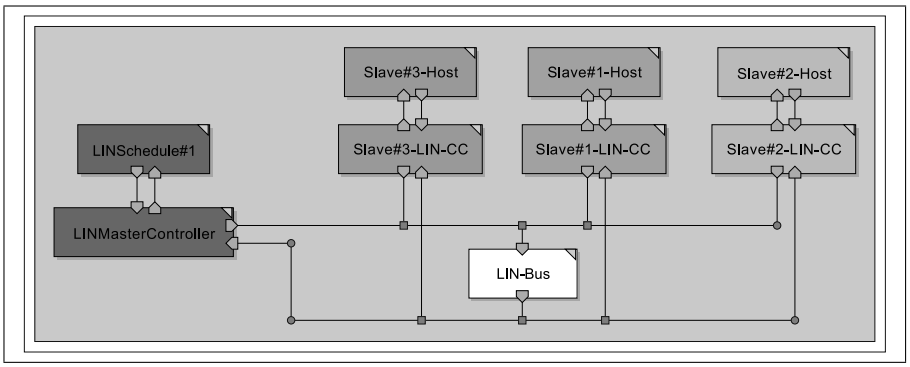


Abbildung A.4.1: Beispiel zur Struktur des LIN-Modells

Der grundsätzliche Aufbau eines LIN-Clusters, unabhängig von einem übergeordneten Bussystem, ist in Abbildung A.4.1 dargestellt. Der dargestellte Cluster besteht dabei aus insgesamt vier Teilnehmern einem Master-Knoten und drei Slave-Knoten. Der Master in diesem Cluster steuert lediglich die Kommunikation und besitzt in diesem Szenario keine eigenständige Applikationsebene, daher ist keine Host-Komponente angeschlossen.

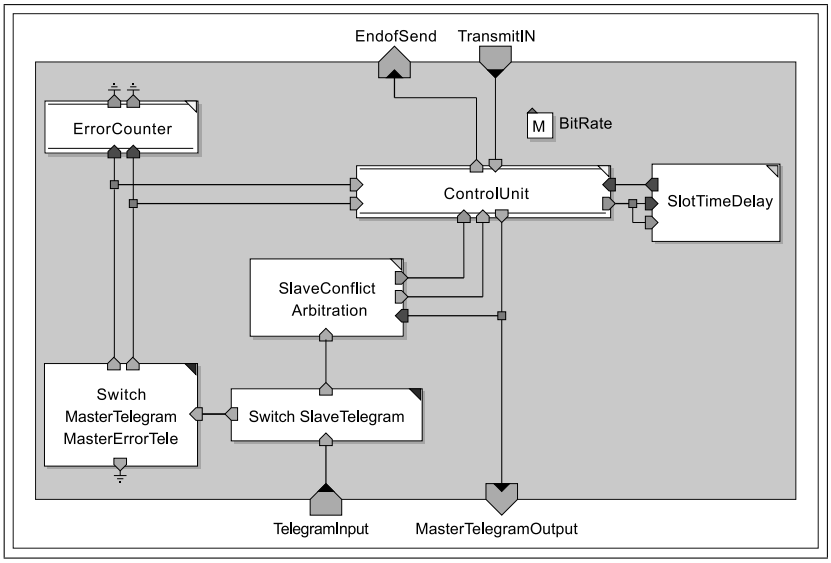


Abbildung A.4.2: Darstellung des Communication Controller (CC) des LIN-Master

Der Master-Knoten wird innerhalb der Modellkomponente aufgeteilt in Master-

Task und Slave-Task, so gibt es zwei Typen von LIN-Communication-Controllern (LINMaster-Controller und Slave-LIN-CC). Während den einzelnen Slave-CCs jeweils eine Host-Komponente zugeordnet ist, wird dem Master-CC lediglich der Schedule zur Steuerung der Kommunikation zugeordnet. Der Communication Controller des LIN-Master ist in Abbildung A.4.2 zu sehen.

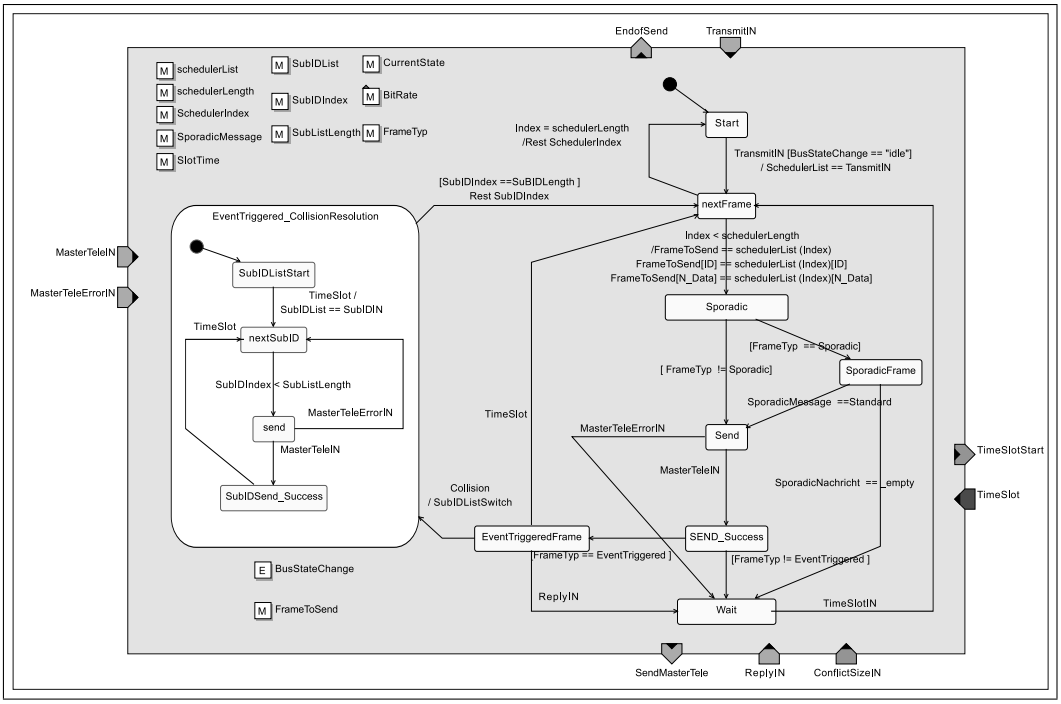


Abbildung A.4.3: FSM der ControlUnit des CC des LIN-Master

Wie bereits die vorhergehenden Modelle werden auch die Komponenten des LIN in die Struktur des Meta-Modell eingeordnet. Dieses ist in Abbildung A.4.4 dargestellt.

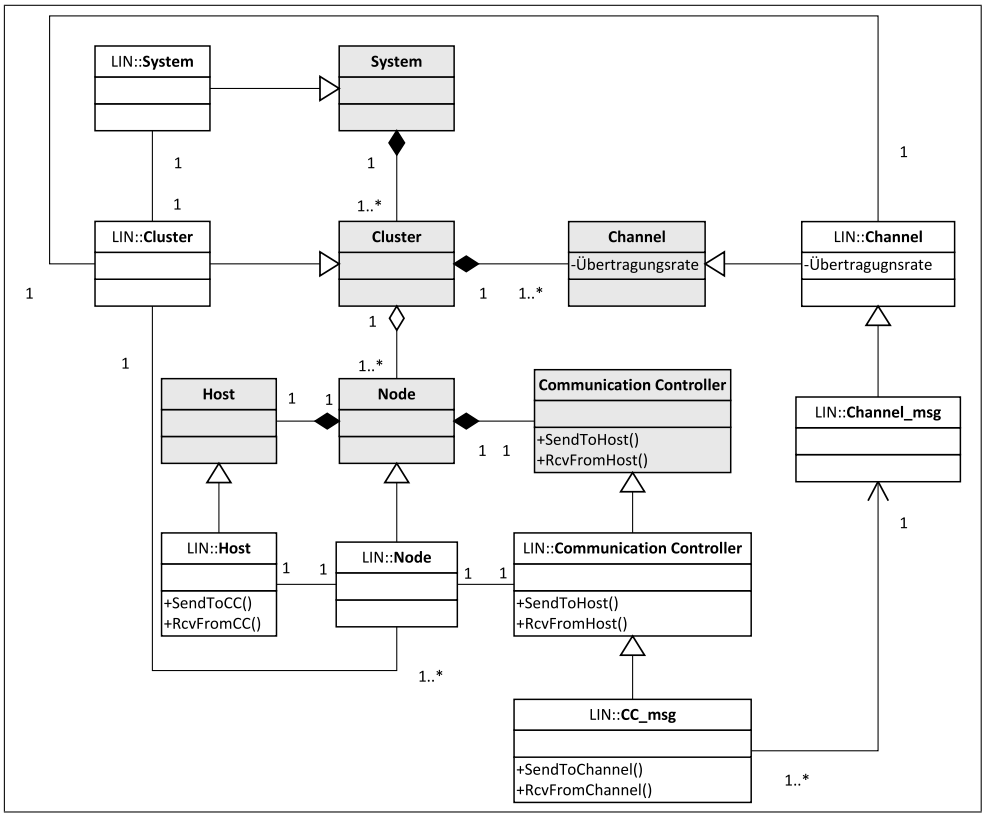


Abbildung A.4.4: Meta-Modell der Modellbibliothek für LIN

A.5 Transformation von FIBEX nach MLD mittels XSLT am Beispiel des CAN-Bus

Das XSLT-Stylesheet verarbeitet ein FIBEX-Dokument, welches die Systembeschreibung eines CAN-Bus enthält. Das System wird in ein MLDesigner-Modell überführt. Es werden MLD-Klassendateien verwendet, welche die entsprechenden Systemelemente bereitstellen. Durch eine Zuordnungsvorschrift (Map-Datei) werden die Zusammenhänge der CAN-Elemente des FIBEX und der korrespondierenden MLD-Klassen definiert, sowie zusätzliche Informationen für das Zielformat bereitgestellt. Weiterführende Informationen zu diesem Stylesheet finden sich in der zugehörigen Dokumentation (Normen Weller, Johannes Klöckner, 2007).

Die Map-Datei trägt den Namen „fib2mldMap.xml“ und befindet sich im selben Verzeichnis wie das Stylesheet. Sie enthält die Abbildungsvorschriften für die CAN-Bus-Elemente, sowie deren Vernetzung in MLDesigner. Zusätzlich können für das MLDesigner Modell Toplevel-Parameter definiert werden, welche von gleichnamigen Toplevel-Memories (falls vorhanden) referenziert werden. Im Folgenden ist als Beispiel eine Map-Datei abgebildet und die einzelnen Elemente werden kurz erläutert:

Listing A.1: Beispiel einer Map-Datei für ein CAN-System

```
1
2 <?xml version="1.0" encoding="ISO-8859-1"?>
3
4 <map>
5    <version>0.3</version>
6    <can>
7       <elements>
8          <channel>CAN/frame_based/CAN_Channel/CAN_Channel.mmk</
       channel>
9          <controller>CAN/frame_based/CAN_Node/CAN_Node.mmk</
       controller>
10      </elements>
11      <wiring>
12         <link element="channel" port="PacketIn" wire="1"/>
13         <link element="channel" port="PacketOut" wire="2"/>
14         <link element="controller" port="PacketOut" wire="1"/>
15         <link element="controller" port="PacketIn" wire="2"/>
16      </wiring>
17      <parameters>
18         <cluster name="BitRate" valueFrom="fx:SPEED" attributes="
       A_CONSTANT|A_SETTABLE" scope="Internal" type="float"/>
19         <cluster name="Beispiel" value="2341" attributes="
       A_CONSTANT|A_SETTABLE" scope="Internal" type="float"/>
20      </parameters>
21   </can>
22 </map>
```

<can> beinhaltet alle Map-Einträge für ein CAN-Bus-System.

<elements> enthält die Elemente des CAN-Bus, welche in der FIBEX-Datei vorkommen und in ein MLDesigner-Modell abgebildet werden. Die Namen dieser Elemente entsprechen den Namen der FIBEX-Elemente in Kleinschreibung. Der Wert der Elemente ist jeweils ein String, welcher den Pfad zu der zugehörigen MLD-Klassendatei darstellt. Der Pfad ist relativ zum MLD-User-Verzeichnis, das MLD-User-Verzeichnis wird dem Stylesheet als Parameter übergeben.

<wiring> enthält die Verbindungsstruktur der unter <elements> angeführten Can-Bus-Elemente. Die Kindelemente <link> bilden (möglichst) jeden Port der oben genannten Can-Bus-Elemente auf einen Draht (Leitung, „wire“) ab.

<parameters> gibt die Parameter für das MLDesigner Modell an.

<cluster> stellt die Parameter dar, welche es für jeden Cluster gibt. Die Parameter werden mit allen in der Map angeführten Attributen, mit Ausnahme von @valueFrom und @value, in das Ergebnisdokument übernommen. Von den eben genannten Attributen kann nur eines exklusiv genutzt werden, andernfalls wird nur das letztgenannte ausgewertet. Dem Attribut @name wird zur eindeutigen Identifikation (automatisch) der Clustername vorangestellt.

valueFrom: gibt an, wo der Wert des Parameters in der FIBEX-Datei zu finden ist.

value: gibt den Wert direkt an.

Es gibt zwei Parameter, die dem Stylesheet übergeben werden können. Dies ist nötig, falls die zu übergebenden Werte vom Default-Wert abweichen.

g_mld_user gibt den Pfad zur MLD-User-Library an [*Default: „~/MLD/“*].

g_mapUri gibt den Namen und Pfad der Map-Datei an (relativ zum Stylesheet oder absolut) [*Default: „fib2mldMap.xml“*].

Die MLDesigner-Modelle (Klassendateien) müssen gewisse Voraussetzungen erfüllen, damit eine Transformation möglich ist. Grundsätzlich gilt, das die Bezeichnungen identisch sein müssen. So muss ein Memory- oder ein Event-Element in den verschiedenen Modellelementen (Klassendateien) gleich benannt werden, wenn dieser auf der Toplevel-Ebene verlinkt werden. Hierzu nachfolgend ein kleines Beispiel.

In der Klassendatei des CAN-Controllers gibt es ein Element <*memory*> namens *„BitRate“* mit der Eigenschaft *„scope=External“* und in der Klassendatei des CAN-Channel gibt es ebenfalls ein Element <*memory*> namens *„BitRate“* mit der Eigenschaft *„scope=External“*; auf der Toplevel-Ebene kann nun ein Element <*memory*> mit dem Namen *„BitRate“* erstellt werden, das von diesen beiden Entitäten referenziert wird.

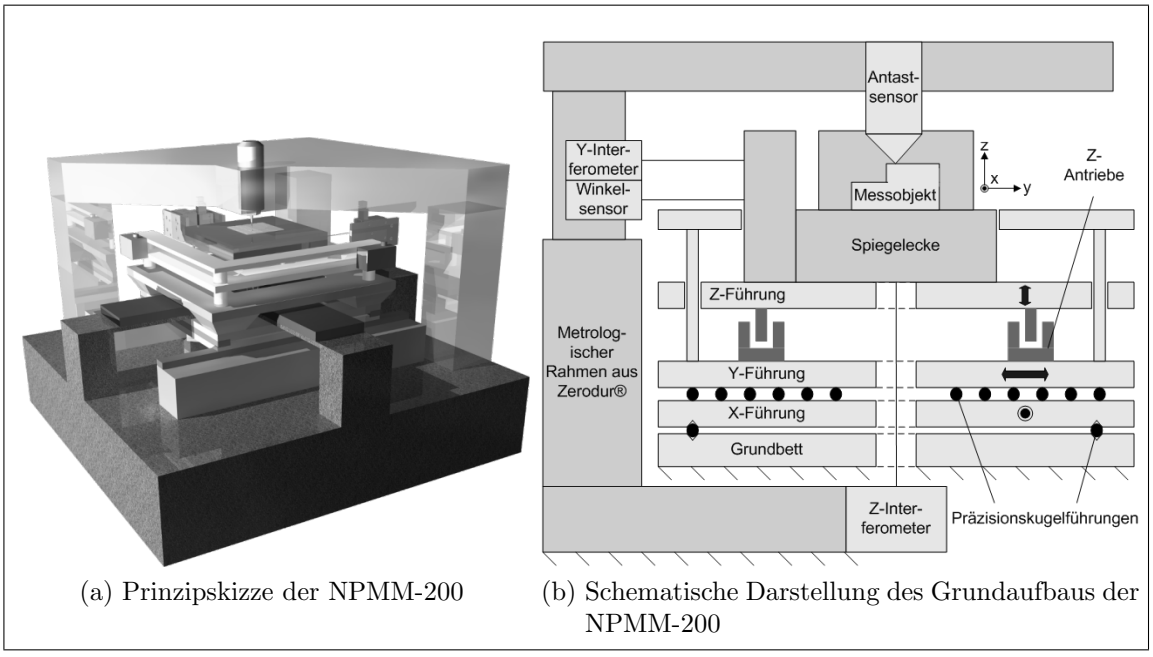
A.6 NPMM-200 Signal- und Datenverarbeitungseinheit: Randbedingungen und umgebender Prozess

Die zentralen Randbedingungen für die Signal- und Datenverarbeitungseinheit (Signal- and Data Processing Unit - SDPU) ergeben sich aus den verwendeten Sensoren und Aktoren innerhalb des mechanischen Aufbaus und des Antriebssystems, den geforderten Taktraten sowie algorithmischen Anforderungen im Bereich Ablaufsteuerung, Trajektorien-Planung, Regelung sowie Vor- und Nachverarbeitung von Signalen und Messwerten.

Detaillierte und weiterführende Informationen zu dem grundlegenden Aufbau von Nanopositionier- und Nanomessmaschinen insbesondere zum Vorgänger der NPMM-200 finden sich in (Hausotte, 2002). Eine umfassendere Betrachtung ist in (Hausotte, 2011) zu finden, hier wird ansatzweise auch die NPMM-200 dargestellt. Eine Betrachtung mit dem Schwerpunkt auf das Regelungskonzept und die Antriebe findet sich in (Zschäck et al., 2013).

A.6.1 Mechanischer Aufbau

Bei dem experimentellen Aufbau handelt es sich um den Prototypen einer Nanopositionier- und Nanomessmaschine mit einem planaren Bewegungsbereich von $200 \times 200 \times 25\ mm^3$ (siehe Abbildung A.6.1a).



(a) Prinzipskizze der NPMM-200

(b) Schematische Darstellung des Grundaufbaus der NPMM-200

Abbildung A.6.1: Prinzipskizze und schematische Darstellung des Grundaufbaus einer NPMM ($200 \times 200 \times 25\ mm^3$) (Zschäck et al., 2013)

Der Aufbau der Maschine kann der schematischen Darstellung des Grundaufbaus

in Abbildung A.6.1b entnommen werden. In diesem Schnitt lassen sich einige Teilkomponenten des Systems erkennen.

Im mittleren Teil der Darstellung findet sich das Messobjekt, welches zum Zweck der Vermessung mit einem Antastsensor (Tastsystem) auf der Spiegelecke positioniert ist. Die Spiegelecke selbst ist aus Zerodur® gefertigt und auf einem dreiachsigen Führungssystem befestigt. Die drei den Interferometern zugewandten Seiten sind jeweils verspiegelt. Die Position der Spiegelecke wird durch stabilisierte HeNe-Laserinterferometer der Firma SIOS GmbH[1] gemessen. Die verwendeten Interferometer erzielen eine Positionsauflösung von $0,08\ nm$. In der Darstellung des Grundaufbaus sind die Interferometer der Z- und der Y-Achse zu erkennen. Unter dem Y-Interferometer ist zudem ein Winkelsensor eingezeichnet, welcher die Drehung (Verkippung) der Spiegelecke um die X-Achse messen kann. Für die beiden anderen Achsen existiert auch jeweils ein solcher Sensor.

Die Längen- und Winkelmesssysteme (Positionsmesssystem) befinden sich an einem metrologischen Rahmen aus Zerodur® und bilden so eine feste Anordnung. Durch die Materialeigenschaften werden thermische Einflüsse auf die Anordnung minimiert. Zudem steht der Gesamtaufbau zur Minimierung des Eintrags von Vibrationen auf einem schwingungsgedämpften Fundament.

Jede der beiden planaren Achsen wird von zwei asynchronen Linearmotoren angetrieben. Die Motoren werden von speziell für diese Anforderungen konstruierten Analogverstärkern mit Strom versorgt. Die positionsabhängige Kommutierung der Linearmotoren erfolgt durch das Echtzeitsystem. Die Analogverstärker werden über 16-Bit Digital/Analog Wandler ($\pm 10\ V$) vom Echtzeitsystem angesteuert. Eine integrierte Stromregelung gewährleistet einen zur Eingangsspannung proportionalen Ausgangsstrom ($\pm 3\ A$). Der aktuell fließende Strom kann über die Analogverstärker messtechnisch erfasst werden.

Im Folgenden werden die drei Teilbereiche Positionsmesssystem (Positioniersystem), Antastsystem (Tastsystem) und Antriebssystem hinsichtlich Sensoren und Aktoren, Taktraten und Algorithmen bzw. Verarbeitung kurz betrachtet.

A.6.1.1 Positionsmesssystem

Innerhalb des Positionsmesssystem werden die positionsabhängigen Messdaten der Maschine erzeugt und verarbeitet. Der grundsätzliche Ablauf der Verarbeitung ist in Abbildung A.6.2 dargestellt. Das System nutzt 18 analoge Signale (Positions- und Winkelinformationen), die von den mechanischen, elektrischen und optischen Systemen abgeleitet werden. Die Signalerfassung erfolgt mit einer Frequenz von $666\frac{2}{3}\ kHz$. Die Erzeugung sowie Vor- und Nachverarbeitung der Messdaten wird mit einer Frequenz von $83\frac{1}{3}\ kHz$ durchgeführt. Die Taktfrequenzen leiten sich dabei von der Leistungsfähigkeit der verfügbaren Hardware-Komponenten ab. Die Online-Korrekturen berücksichtigen verschiedene Einflussfaktoren: systematische Nichtlinearitäten (Justage und ähnliche Fehlerquellen der Interferometeroptik und -elektronik), Umweltfaktoren (z.B. Temperatur, Luftdruck und -feuchte sowie CO2-Gehalt), Nichtorthogonalität der Winkel zwischen den einzelnen Spiegeln und Unebenheiten in den Spiegeln,

[1] http://www.sios.de/

Abbe-Fehler zwischen Tastsystem und Messobjekt.

Die zentrale Herausforderung ist die Messwertverarbeitung mit einer Zykluszeit von 12 μs, welche nur geringe Latenzen toleriert. Innerhalb des Zyklus müssen komplexe Algorithmen mit Fließkomma-Arithmetik (single- und double Precision) in der Vor- und Nachverarbeitung durchgeführt werden. Die Nachverarbeitung arbeitet auf dem gesamten Messdatensatz, der hierfür notwendige in-cycle Datenaustausch (Datenaustausch innerhalb des Verarbeitungszyklus) erfordert eine leistungsfähige Kommunikation.

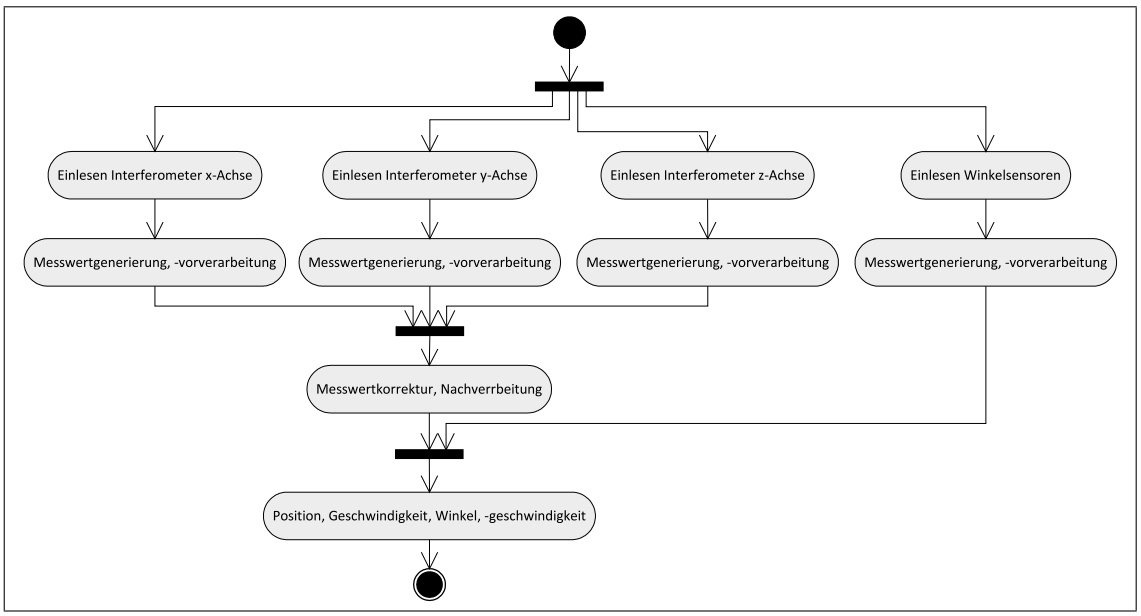


Abbildung A.6.2: Prinzipieller Ablauf Positioniersystem

A.6.1.2 Antastsystem

Das konkrete Antastsystem ist abhängig vom verwendeten (verbauten) Antastsensor. Allgemein stehen für die Interaktion mit einem Tastsystem acht analoge Eingangssignale und drei Ausgangssignale sowie vier interferometrische Längenwerte zur Verfügung. Die Verarbeitungsalgorithmen sind im Wesentlichen Polynomberechnungen mit Fließkomma-Arithmetik (single precision), welche für ein konkretes Tastsystem jeweils parametrisiert werden müssen. Die Verarbeitung erfolgt im Messtakt mit $83,3\ kHz$. Der grundsätzliche Ablauf ist in Abbildung A.6.3 dargestellt

A.6.1.3 Antriebssystem

Zum Antriebssystem der Maschine werden an dieser Stelle das Regelungssystem sowie die Ausgabe der Stellgrößen gezählt. Die Generierung der Sollwerte im Rahmen der Bahnplanung (Trajektorien-Generierung) wird an dieser Stelle nicht näher betrachtet. Das Regelungssystem der NPMM-200 soll dafür sorgen, dass der vorgegebenen

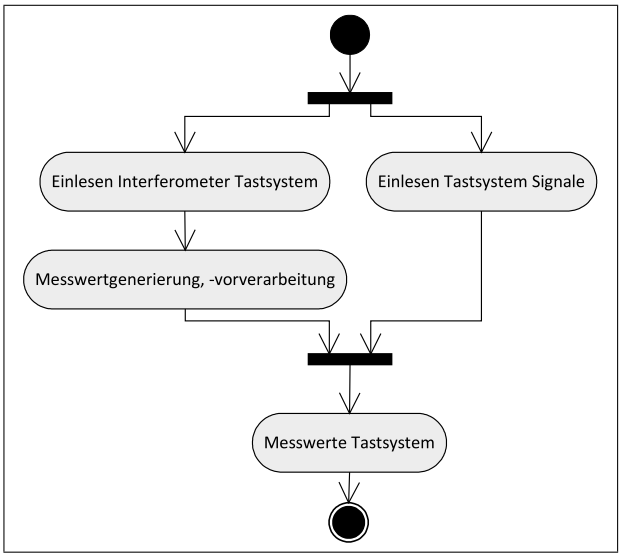


Abbildung A.6.3: Prinzipieller Ablauf Antastsystem

Trajektorie exakt und ohne Schleppfehler gefolgt wird. Dafür ist eine Kompensation auftretender Reibungskräfte sowie externer Störungen wie Vibrationen und Schallwellen notwendig. Die schematische Darstellung des verwendeten Regelungssystems ist in Abbildung A.6.4 dargestellt. Eine detaillierte Beschreibung des Regelungssystem findet sich in (Zschäck et al., 2013).

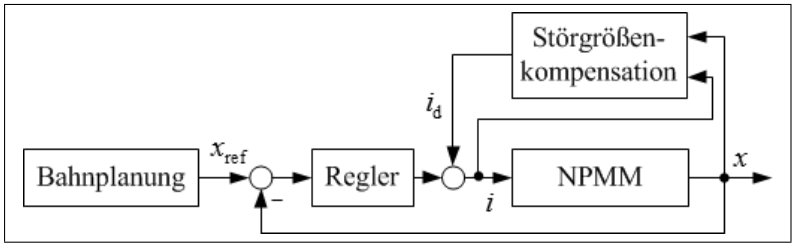


Abbildung A.6.4: Schematische Darstellung des Regelkreises für eine Achse

Für die Ansteuerung der Motoren der Achsen x und y müssen jeweils vier Stellgrößen ausgegeben werden und für die z-Achse drei Stellgrößen. Für Kontrollzwecke können die jeweiligen Stellgrößen zurückgelesen werden. Zentrale Herausforderung ist die Berechnung der Stellgrößen innerhalb eines Regeltaktes von $8,3\ kHz$. Der abstrakte funktionelle Ablauf ist in Abbildung A.6.5 dargestellt.

A.6.2 Signal- und Datenverarbeitungseinheit

Für die Realisierung der Signal- und Datenverarbeitungseinheit stehen standardisierte Hardware-Komponenten zur Verfügung. Dieses sind PXI-Systeme bestehend aus Echtzeit-Controller des Typs PXI-8108-RT und drei verschiedenen Typen von FPGA-Modulen. Die FPGA-Module vom Typ PXI-7853R und PXI-7854R haben je-

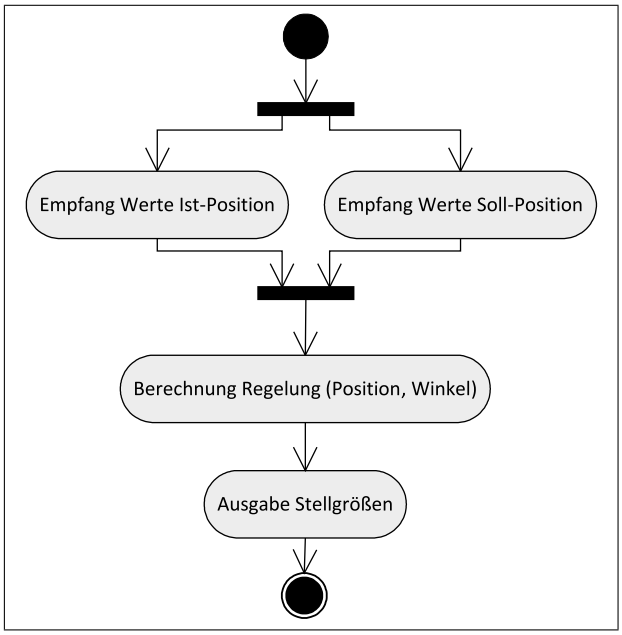


Abbildung A.6.5: Prinzipieller Ablauf Regelungssystem

weils 8 analoge Eingangssignale, 8 analoge Ausgangssignale und 96 digitale Signale (Ein- oder Ausgänge), die FPGA-Module vom Typ PXI-7813R besitzen 160 digitale Ein-/Ausgangssignale. Der Datenaustausch zwischen den FPGA-Modulen als Schnittstelle zum Prozess und den RT-Controllern erfolgt grundsätzlich über PCI-Backplane innerhalb des PXI-Chassis. Der prinzipielle Aufbau eines PXI-Systems sowie die verfügbaren FPGA-Module werden im Anhang A.8 erläutert.

Die Signal- und Datenverarbeitungseinheit der NPMM-200 baut auf standardisierten Hardware-Komponenten auf, stößt aber in Leistungsbereiche vor, für die diese einzelnen Komponenten nicht konzipiert sind. Dies zeigen die im Datenblatt für den Echtzeitcontroller PXI-8110 RT (National Instruments, 2014a) angegeben Leistungsparameter für die Verarbeitung, welche in Tabelle A.2 aufgeführt sind.

Tabelle A.2: Real-Time Performance (Maximum Loop-Rate PXI-8106 RT, PXI-8108 RT und PXI-8110 RT) (National Instruments, 2014a)

				Loop Rate (kHz)		
Benchmark	Processing	Channels	DAQ I/O Mode	PXI 8106 RT	PXI 8108 RT	PXI 8110 RT
Analog I/O	PID	1	Polling	86	136	129
Analog I/O	PID	1	Interrupt	35	50	54
Analog I/O	PID	4	Polling	51	77	75
Analog I/O	PID	4	Interrupt	33	40	41
Analog I/O	PID	16	Polling	26	31	31
Analog I/O	PID	16	Interrupt	16	24	24

Für dieses komplexe verteilte System zur Echtzeitverarbeitung ergibt sich als zen-

trale Herausforderung die Erfüllung der zeitlichen Anforderungen durch die hohen Taktraten. Direkt beeinflusst wird dies durch die Verteilung der Funktionen bzw. der Systempartitionierung, die Berechnung der Algorithmen sowie durch den Datenaustausch und die Kommunikation innerhalb des verteilten Systems. Insbesondere durch die Möglichkeit der prozessnahen Verarbeitung innerhalb der FPGAs ergeben sich zahlreiche Varianten für die Realisierung des Systems.

A.7 NPMM-200 Signal- und Datenverarbeitungseinheit: Zeitverhalten Prototyp

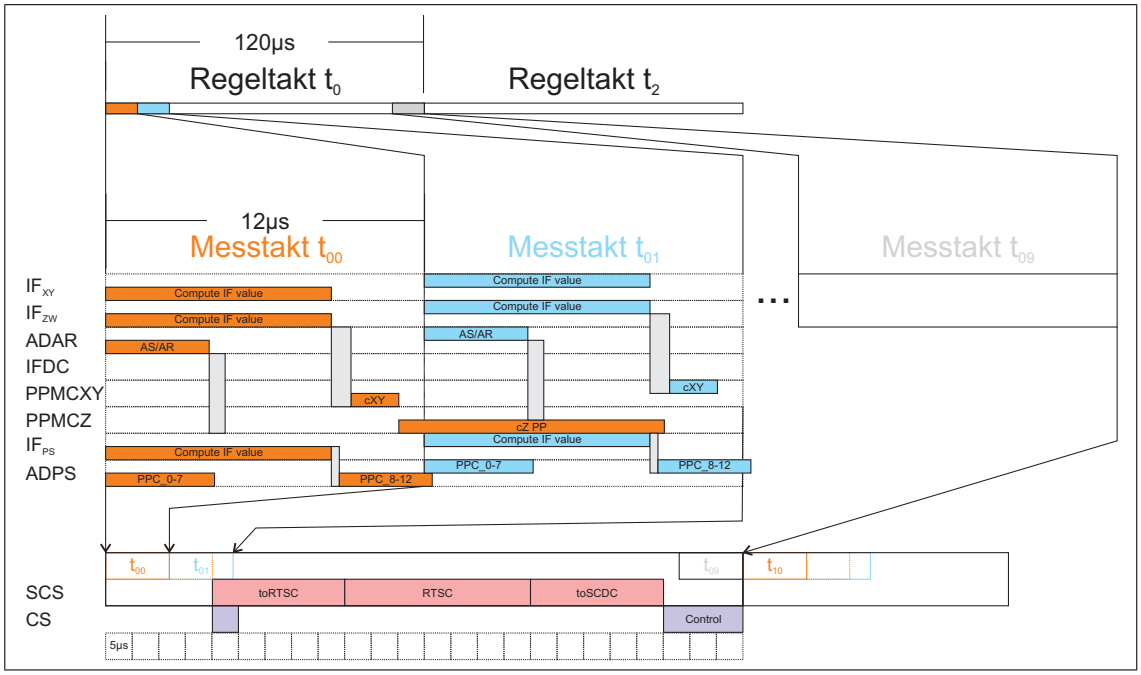


Abbildung A.7.1: Schematische Darstellung des Zeitverhaltens des Prototypen der Signal- und Datenverarbeitungseinheit NPMM-200

A.8 PXI- und PXI Express-Architektur

Die PXI-Spezifikation (PCI[2] eXtensions for Instrumentation) ist ein von National Instruments auf Basis des PCI-Standards entwickelter offener Industriestandard für eine modular aufgebaute, PC-basierte Plattform für Mess- und Automatisierungssysteme. Der Standard wird von der PXI Systems Alliance verwaltet. An diesem Bündnis

[2]Peripheral Component Interconnect

sind über 65 Firmen beteiligt, welche basierend auf dem PXI-Standard Produkte entwickeln und vertreiben, die Kompatibilität der Systemmodule wird dabei über die PXISA garantiert.(National Instruments, 2014f; PXI Systems Alliance, 2004)

PXI-Systeme setzen sich wie in Abbildung A.8.1 dargestellt aus drei Grundkomponenten zusammen: dem Chassis mit Backplane, dem System-Controller und diversen Peripheriemodulen.

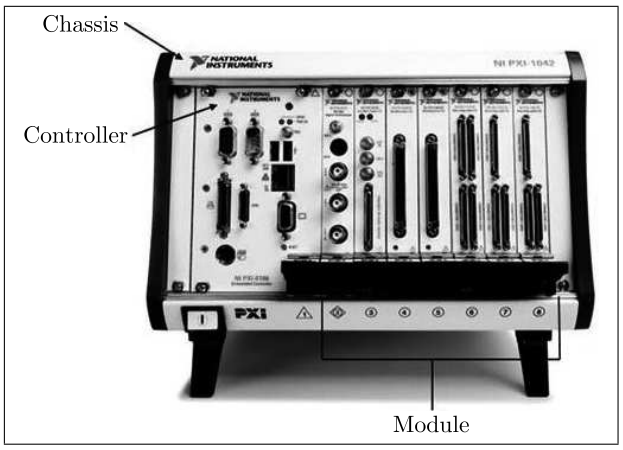


Abbildung A.8.1: Grundaufbau eines PXI-Systems (National Instruments, 2014f)

A.8.1 Architektur

A.8.1.1 PXI-Architektur

In Abbildung A.8.2 ist der schematische Aufbau eines PXI-Systems dargestellt. Das PXI-Chassis beinhaltet die dargestellten Komponenten einen System-Controller, ein optionales Trigger-Modul, div. Module sowie das Kernstück die PXI-Backplane. Die PXI-Backplane beinhaltet die Busleitungen (*PCI-Bus*, *Star Trigger Bus*, *PXI Trigger Bus* und *Local Bus*), den PXI-Referenztaktgeber (10 MHz Reference Clock) sowie die Modul-Schnittstellen zu den Modulen und unterstützt 4 bis 18 Steckplätze. Dabei ist der erste Steckplatz fest dem PXI-Controller und der zweite optional einem Trigger-Modul zugeordnet.

Der PXI-Trigger Bus bietet 8 Datenleitungen, die flexibel für die Synchronisation genutzt werden können. Eine schnellere und präzisere Übertragung von Trigger-Signalen ist über den *Star-Trigger Bus* möglich. Der Star-Trigger-Controller ist mit 13 bidirektionalen Busleitungen mit den peripheren Modulen verbunden. Mit Hilfe des *Local Bus* (13 Busleitungen) können Daten zwischen benachbarten Modulen ausgetauscht werden (falls die PXI-Module den *Local Bus* unterstützen).

A.8.1.2 PXIe-Architektur

Im Jahre 2005 wurde die PXI Express Spezifikation (PXI Systems Alliance, 2005) verabschiedet, die schematische Struktur ist in Abbildung A.8.3 dargestellt. Die Kom-

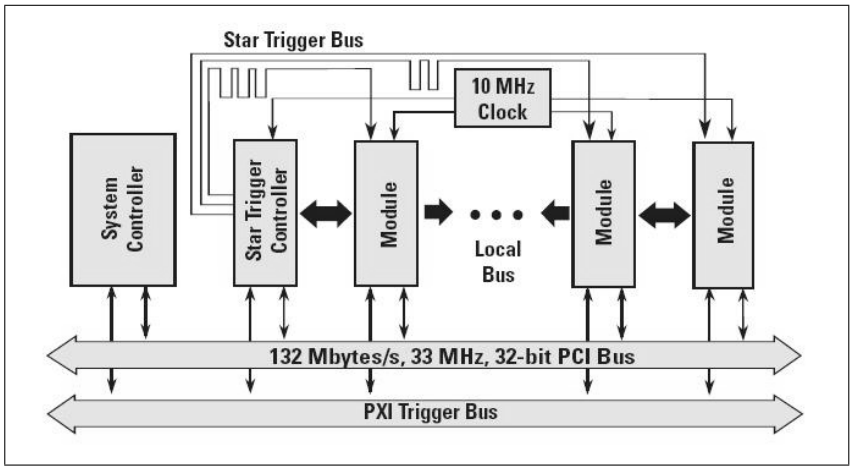


Abbildung A.8.2: Schematische Darstellung eines PXI Systems (National Instruments, 2014f)

munikationswege wurden erweitert und verbessert. Es kommen eine differenzielle 100 MHz Systemuhr, Differenzsignalisierung und differenzieller Star-Trigger zum Einsatz. Für die Anbindung der Module gibt es drei unterschiedliche Slot-Typen: Der reine PXI-Express-Slot, der hybride Slot zur Aufnahme von PXI- und PXIe-Modulen und der reine PXI-Slot. Für den Entwurf und die Entwicklung eines Systems ist die Entscheidung welches Modul welchen Slot erhält wichtig.

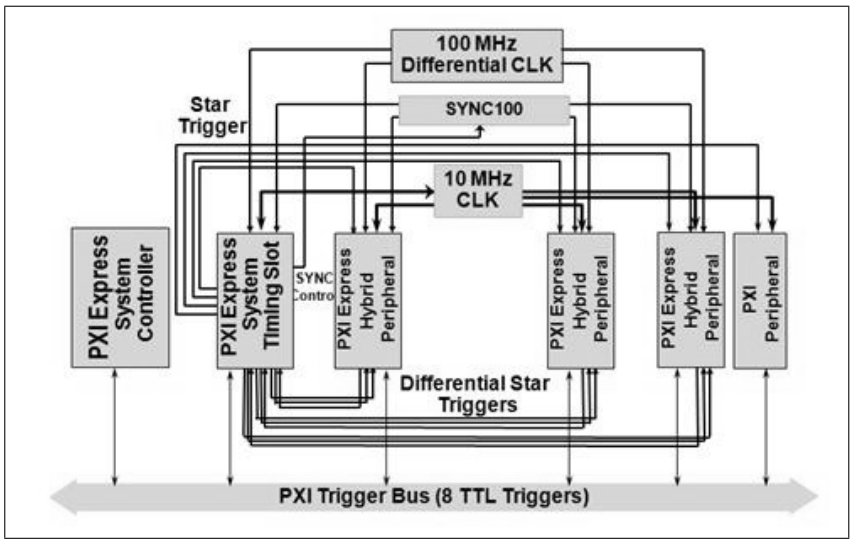


Abbildung A.8.3: Schematische Darstellung eines PXI-Express Systems (National Instruments, 2014f)

A.8.2 Komponenten eines PXI-Systems

A.8.2.1 Chassis

Das PXI-Chassis gewährleistet den modularen System-Charakter und besitzt vier bis achtzehn Steckplätze für verschiedene Peripheriemodule. Das Chassis beinhaltet die Backplane mit PCI-Bus und zusätzlichen Timing- und Trigger-Bussen zur Synchronisation. Die Einführung von PXI Express erlaubt die Nutzung der Bandbreitenvorteile von PCI Express in Kombination mit der Kompatibilität zu PCI.

A.8.2.2 Controller

Abbildung A.8.4: PXI-Controller (National Instruments, 2014b)

Der PXI-Controller (Abbildung A.8.4) befindet sich im Chassis im ersten Steckplatz. Für die Systemsteuerung gibt es je nach konkretem System unterschiedliche Varianten: Remote-Systemsteuerung oder Embedded Controller mit einem Windows und/oder einem Echtzeitbetriebssystem. Für die Systementwicklung im Rahmen des Projektes stehen PC-basierten Realtime-Controller vom Typ PXI-8108-RT zur Verfügung (Dual-Core-Prozessor Intel Core 2 Duo T9400 mit 2,53 GHz, 2 GB DDR2-RAM mit 800 MHz, Ethernet, USB usw.) (National Instruments, 2009b).

A.8.2.3 Module

Nach den beiden Basiselementen Chassis und Controller spielen die flexibel kombinierbaren Module eine zentrale Rolle, da mit diesen die eigentliche Funktionalität eines PXI-Systems charakterisiert wird und die Anbindung an die Peripherie erfolgt. Es existieren für die verschiedensten Anwendungsbereiche Module, beispielsweise Module zur Motorensteuerung, Signalerzeugung, Signalkonditionierung, Bildverarbeitung, Zähler, Kommunikation, Datenprotokollierung sowie Schaltmodule.

Für die Systementwicklung im Rahmen des Projektes werden drei FPGA-Module eingesetzt: PXI-7813R, PXI-7853R und PXI-7854R (National Instruments, 2009a). Diese haben den Vorteil, dass sie mit ihren analogen und digitalen Ein- und Ausgängen zum einen die Schnittstelle zur Maschine realisieren und zum anderen einen frei programmierbaren FPGA beinhalten, welcher zur prozessnahen Verarbeitung verwendet werden kann. Die ausgewählten FPGA-Module werden in der Tabelle A.3 kurz charakterisiert.

Tabelle A.3: FPGA-Module in der NPMM-200

	PXI-7853R	PXI-7854R	PXI-7813R
	 (National Instruments, 2014d)	 (National Instruments, 2014e)	 (National Instruments, 2014c)
Connector 0	8 analog In (750 kHz, 16 Bit, $\pm 10V$), 8 analog Out (750 kHz, 16 Bit, $\pm 10V$), 16 digitale I/O (40 MHz)		40 digitale I/O (40 MHz)
Connector 1	40 digitale I/O (40 MHz)		
Connector 2	40 digitale I/O (40 MHz)		
Connector 3	nicht vorhanden		40 digitale I/O (40 MHz)
FPGA	Virtex-5-LX85	Virtex-5-LX110	Virtex-II V3000

Literatur

Adams, M., Kühn, W., Stör, T. & Zelm, M. (2007). DIN EN 62264 . Die neue Norm zur Interoperabilität von Produktion und Unternehmensführung - Teil 1. *atp - Automatisierungstechnische Praxis*, *49* (5), 52–57.

Al Ali, K. (2011). *Modellierung und Verifikation von verteilten/parallelen Informationssystemen* (Dissertation). Ilmenau University of Technology.

Alvarez, J., de la Camara, P., Martinez, J., Merino, P., Perez, F. & Morillo, V. (2006). An SDL Implementation of the UMTS Radio Resource Control Protocol Oriented to Conformance Testing. In *Leveraging Applications of Formal Methods, Verification and Validation, 2006. ISoLA 2006. Second International Symposium on* (S. 397-401).

Angermann, A., Beuschel, M., Rau, M. & Wohlfahrt, U. (2007). *Matlab - Simulink - Stateflow: Grundlagen, Toolboxen, Beispiele* (5. Aufl.). Oldenbourg.

ASAM. (2007). *FIBEX - Field Bus Exchange Format, Version 2.0.1* [Specification].

AUTOSAR GbR. (2008a). *Requirements on Gateway, Release 3.1, Version 2.0.5* [Specification].

AUTOSAR GbR. (2008b). *Specification of PDU Router, Release 3.0, Version 2.2.2* [Specification].

Bago, M., Perić, N. & Marijan, S. (2008). Modeling Bus Communication Protocols Using Timed Colored Petri Nets - The Controller Area Network Example. In *Ninth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools* (S. 103-121).

Baldwin, P., Kohli, S., Lee, E. A., Liu, X. & Zhao, Y. (2004). Modeling of Sensor Nets in Ptolemy II. In *Proceedings of the 3rd International Symposium on Information Processing in Sensor Networks* (S. 359–368). New York, NY, USA: ACM.

Balzer, F., Klöckner, J., Zschäck, S. & Percle, B. (2012). *Building a Nanomeasuring Machine Using LabVIEW and NI PXI* [Graphical System Design Achievement Awards (Winner in Advanced Control Systems)]. Austin, Texas. Zugriff am 08.01.2015 auf `http://sine.ni.com/cs/app/doc/p/id/cs-14759`

Baumann, T. (2009). *Beiträge zur Automatisierung der frühen Entwurfsphasen verteilter Systeme* (Dissertation). Technische Universität Ilmenau.

Baumann, T. and Hauguth, M. and Salzwedel, H. (2007). Overcoming the Gap between Design at Electronic System Level (ESL) and Implementation for Networked Electronics. In *Western MultiConference on Modeling & Simulation.*

Baumgarten, U. & Siegert, H.-J. (2007). *Betriebssysteme: Eine Einführung* (6., überarb., aktualisierte und erw. Aufl.). München und Wien: Oldenbourg.

BMBF. (2012). *Zukunftsbild Industrie 4.0* (Broschüre). Bonn: Bundesministerium für Bildung und Forschung (BMBF). Zugriff am 08.01.2015 auf `http://www.bmbf.de/pubRD/Zukunftsbild_Industrie_40.pdf`

Boller, R. (2010). *Konzeption und Umsetzung einer Kommunikationsinfrastruktur für ein komplexes verteiltes System basierend auf PXI-Systemen* (Diplomarbeit). Technische Universität Ilmenau.

Buck, J., Ha, S., Lee, E. A. & Messerschmitt, D. G. (1991). Ptolemy: A Mixed-Paradigm Simulation/Prototyping Platform in C++. In *Proceedings of the C++ At Work Conference.*

Cassandras, C. G. & Lafortune, S. (2008). *Introduction to Discrete Event Systems.* Boston and MA: Springer Science+Business Media LLC.

Comer, D. (2004). *Computernetzwerke und Internets: mit Internet-Anwendungen.* Pearson Studium.

Conrads, D. (1996). *Datenkommunikation* (3. Aufl.). Vieweg Verlag.

Deutsche Forschungsgemeinschaft. (2014). Neue Großgeräteinitiative: „Nanopositionier- und Messmaschinen". *Information für die Wissenschaft* (06). Zugriff am 08.01.2015 auf `http://www.dfg.de/foerderung/info_wissenschaft/2014/info_wissenschaft_14_06/index.html`

Eker, J., Janneck, J., Lee, E., Jie, L., Xiaojun, L., Ludvig, J., ... Yuhong, X. (2003). Taming heterogeneity - the Ptolemy approach. *Proceedings of the IEEE*, *91* (1), 127-144.

Etschberger, K. (2002). *Controller-Area-Network* (3. Aufl.). Hanser Verlag.

FlexRay Consortium. (2005a). *FlexRay Communications Systems - Electrical Physical Layer Specification Version 2.1* [Specification].

FlexRay Consortium. (2005b). *FlexRay Communications Systems - Protocol Specification Version 2.1* [Specification].

Francis Rumsey, J. W. (1995). *The digital interface handbook.* Focal Press. (2.Auflage / ISBN: 0-240-51396-7)

Führer, T., Müller, B., Dieterle, W., Hartwich, F. & Hugel, R. (2000). Time-triggered Communication on CAN (Time-triggered CAN-TTCAN). In *Proceedings - 7th International CAN Conference.* Erlangen, Germany: CAN in Automation GmbH.

Furrer, F. J. (1981). *Fehlerkorrigierende Block-Codierung für die Datenübertragung* (Bd. 36). Basel and Boston and Stuttgart: Birkhäuser Verlag.

Georgi, W. & Metin, E. (2009). *Einführung in LabVIEW: mit 146 Aufgaben.* Fachbuchverl. Leipzig im Carl-Hanser-Verlag.

Gravano, S. (2001). *Introduction to error control codes* (Bd. 9). Oxford: Oxford Univ. Press.

Gräbe, T. (2007). *Erstellung eines grafischen Monitor-Programmes für das Monitoring einer FlexRay-Bus-Simulation* (Bachelorarbeit). Fachhochschule Erfurt.

Gruhler, G. (2001). *Feldbusse und Geräte-Kommunikationssysteme.* Poing: Franzis.

Grzemba, A. (2007). *MOST: Das Multimedia-Bussystem für den Einsatz im Automobil* (Bd. Bd. 2). Poing: Franzis.

Grzemba, A. & Wense, H.-C. v. d. (2005). *LIN-Bus: Systeme, Protokolle, Tests von LIN-Systemen, Tools, Hardware, Applikationen.* Poing: Franzis.

Harel, D. (1986). *Statecharts: A visual formalism for complex systems* (Bd. CS86-02). Rehovot and Israel: Weizmann Institute of Science, Dept. of Computer Science.

Haußner, C., Elger, J. & Trautmann, A. (2010). Zusammenfassung und Fazit: Das Internet der Dinge als neues Vorgehensmodell. In W. Günthner & M. Hompel (Hrsg.), *Internet der Dinge in der Intralogistik* (S. 249-255). Springer Berlin Heidelberg.

Hausdörfer, R. (2009). *Entwurf, Realisierung und Untersuchung einer leistungsfähigen Kommunikationsschnittstelle zum Austausch von Daten zwischen PXI - Systemen unter Nutzung von FPGA - Technologie* (Diplomarbeit). Technische Universität Ilmenau.

Hausotte, T. (2002). *Nanopositionier- und Nanomessmaschine* (Dissertation). Technische Universität Ilmenau, Ilmenau.

Hausotte, T. (2011). *Nanopositionier- und Nanomessmaschinen: Geräte für hochpräzise makro- bis nanoskalige Oberflächen- und Koordinatenmessungen* (1. Aufl.). Berlin: Pro Business. (Techn. Univ., Habil.-Schr.–Ilmenau, 2011)

Hedenetz, B. & Belschner, R. (1998). *Brake-by-wire Without Mechanical Backup by Using a TTP-communication Network*. Society of Automotive Engineers.

Heinecke, H., Schedl, A., Berwanger, J., Peller, M., Nieten, V., Belschner, R., ... Bracklo, C. (2002). FlexRay - ein Kommunikationssystem für das Automobil der Zukunft. *Elektronik Automotive* (September).

Henzinger, T. A. & Sifakis, J. (2007). The Discipline of Embedded Systems Design. *Computer*, *40* (10), 32–40.

Hogrefe, D. (1989). *Estelle, LOTOS und SDL: Standard - Spezifikationssprachen für verteilte Systeme*. Springer.

Hohmann, T. (2007). *Modellierung des Kommunikationsprotokolls FlexRay auf Verhaltensebene* (Diplomarbeit). Technische Universität Ilmenau.

Homann, M. (2005). *OSEK: Betriebssystem-Standard für Automotive und Embedded Systems* (2., überarb. Aufl Aufl.). Bonn: mitp-Verl.

Hopcroft, J. & Ullman, J. (1990). *Einführung in die Automatentheorie, formale Sprachen und Komplexitätstheorie*. Addison-Wesley.

Hörner, H. (2007). Das universelle Gateway-Steuergerät. *Elektronik automotive*. (Sonderausgabe AUTOSAR)

Huang, W. (2001). *Estelle verification of ATM available bit rate (ABR) control protocol* (Master Report). Concordia University.

Huang, Y. (2008). *Modellierung von Gateway-Funktionen zur Analyse der Kommunikation in heterogenen, eingebetteten Systemen* (Diplomarbeit). Technische Universität Ilmenau.

Information Technology — Open Systems Interconnection — Basic Reference Model: The Basic Model (ISO/IEC Nr. 7498-1:1994). (1994). Geneva, Switzerland: ISO. Also published as ITU-T Recommendation X.200.

ITU-T. (2002). *ITU-T Recomondation Z.100 (08/02): Specification and Description Language (SDL)* [Specification].

Jensen, J. C., Chang, D. & Lee, E. A. (2011). A Model-Based Design Methodology for Cyber-Physical Systems. In *Wireless Communications and Mobile Computing Conference (IWCMC), 2011 7th International* (S. 1666 - 1671).

Jentzsch, H. (2003). *Modellierung von PDV-Kommunikationssystemen* (Diplomarbeit). Technische Universität Ilmenau.

Kagermann, H., Wahlster, W. & Helbig, J. (2012). *Umsetzungsempfehlungen für das Zukunftsprojekt Industrie 4.0* (Abschlussbericht des Arbeitskreises Industrie 4.0). Berlin: Forschungsunion im Stifterverband für die Deutsche Wissenschaft.

Kappert, J. (2008). *Modellierung von Echtzeitbetriebssystemen zur Unterstützung des Entwurfs und der Analyse eines Gesamtsystems* (Diplomarbeit). Technische Universität Ilmenau.

Kindel, O. & Friedrich, M. (2009). *Softwareentwicklung mit autosar: Grundlagen, engineering, management in der praxis*. Heidelberg: dpunkt.

Klöckner, J., Köhler, S. & Fengler, W. (2008). Model based Design of Networked

Embedded Systems - A Modeling Approach using FlexRay as an Example. In J. Filipe, J. Andrade-Cetto & J.-L. Ferrier (Hrsg.), *ICINCO-SPSMC* (S. 253-259). INSTICC Press.

Klöckner, J., Müller, M., Fengler, W. & Huang, Y. (2009). Modeling Approach for Networked Embedded Systems with Heterogeneous Communication - Modeling Common Gateway Functionalities for Interconnection. In J. Filipe, J. Andrade-Cetto & J.-L. Ferrier (Hrsg.), *ICINCO-SPSMC* (S. 230-233). INSTICC Press.

Knorr, R. (1994). *Spezifikation, Verifikation, Leistungsbewertung und Implementierung von Kommunikationsprotokollen mit hierarchischen High-Level-Netzen.* Shaker.

Kopetz, H. (2001). *A Comparison of TTP/C and FlexRay* (Bericht). Wien, Austria: Technische Universität Wien.

Kopetz, H. & Bauer, G. (2003). The time-triggered architecture. *Proceedings of the IEEE*, *91* (1), 112-126. doi: 10.1109/JPROC.2002.805821

Koshy, T. (2004). *Discrete mathematics with applications.* Elsevier Science.

Krahn, A. (2010). *Intermodulkommunikation in einem Multi-FPGA-System SHIVA - Shared memory Interface for Versatile Application* (Bachelorarbeit). Technische Universität Ilmenau.

Krasner, J. (2004). Model-based design and beyond: Solutions for today's embedded systems requirements. *Analyst report, American Technology International* (January), 1–12.

Kuster, J. (2011). *Handbuch Projektmanagement* (3., erw. Aufl Aufl.). Berlin u.a: Springer.

Lakos, C., Lamp, J., Keen, C. & Marriott, B. (1995). Modelling Network Protocols with Object Petri Nets. In *Proceedings of Workshop on Petri Netz applied to Protocols* (S. 31–42). Springer-Verlag.

Larsen, R. (2010). *LabVIEW for Engineers.* Prentice Hall/Pearson.

Lee, E. A., Hylands, C., Janneck, J., Davis II, J., Liu, J., Liu, X., ... Whitaker, P. (2001). *Overview of the Ptolemy Project* (Bericht Nr. UCB/ERL M01/11). EECS Department, University of California, Berkeley. Zugriff am 08.01.2015 auf `http://www.eecs.berkeley.edu/Pubs/TechRpts/2001/ERL-01-11.pdf`

Leen, G. & Heffernan, D. (2002). TTCAN: a new time-triggered controller area network. *Microprocessors and Microsystems*, *26* (2), 77 - 94.

Liebezeit, T., Zerbe, V. & Löffler, T. (2003). A Framework for Mission Level Design. In *12th IASTED International Conference on Applied Simulation and Modeling (ASM 2003), 03.-05. September 2003, Marbella, Spanien* (S. 616-621). ACTA Press.

LIN Consortium. (2010). *LIN Specification Package Revision 2.2A* [Specification].

Lisner, J. & Kessler, P. (2001). Entwurf eines TTP/C-Feldbusmodells mit der Spezifikationssprache SDL. *Technischer Bericht.*

Liu, B. (2010). *Modellierung von Kommunikationsarchitekturen am Beispiel von PCI und PCI-Express* (Diplomarbeit). Technische Universität Ilmenau.

Lorenz, T. (2008). *Advanced Gateways in Automotive Applications* (Dissertation). Technische Universität Berlin.

Lupini, C. A. (2001). Multiplex bus progression. In *SAE World Congress, 2001, In-Vehicle Networks, Safety Critical Systems, Accelerated Testing, and Reliability,*

2001 (S. 11–20). Warrendale: Society of Automotive Engineers (SAE).

Lupini, C. A. (2003). Multiplex bus progression 2003. In *SAE World Congress, 2003, In-Vehicle Networks, Safety Critical Systems, Accelerated Testing, and Reliability, 2003* (Bd. 1783, S. 43–51). Warrendale: Society of Automotive Engineers (SAE).

mathworks. (2014). *Communication Protocol Modeling in an Ethernet LAN* [Example]. Zugriff am 08.01.2015 auf http://www.mathworks.de/de/help/simevents/examples/communication-protocol-modeling-in-an-ethernet-lan.html

Mikroelektronik, T. (2006). *Flexconfig user manual* [User Manual].

MLDesign Technologies, Inc. (2007). *MLDesigner Documentation* [Handbuch].

Motorola, Inc. (2004). *SPI Block Guide V03.06* [Manual].

Müller, M. (2013). *Beitrag zum modellbasierten Entwurf eingebetteter Systeme : Komponentenbasierte Modellierungsansätze für verteilte rekonfigurierbare Plattformen* (Dissertation). Technische Universität Ilmenau, Ilmenau.

Müller, M., Brandel, O., Krahn, A. & Fengler, W. (2011). Shared-memory communication in distributed SoCs on multi-FPGA systems. In *Proceedings EdaWorkshop 11 : Dresden, May 10 - 12, 2011* (S. 57–62). Berlin [u.a.]: VDE-Verl.

Müller, M., Klöckner, J. & Fengler, W. (2011). Bit accurate timing analysis on a frame based CAN model. In *4th IFAC Workshop on Discrete-Event System Design 2009 : Gandia, Spain, 6 - 8 October 2009 ; [DESDes]* (S. 114–119). Red Hook and NY: Curran.

Müller, M., Klöckner, J., Gushchina, I., Pacholik, A., Fengler, W. & Amthor, A. (2013). Performance evaluation of platform-specific implementations of numerically complex control designs for nano-positioning applications. *IJES*, *5* (1/2), 95-105.

Müller, M., Schwannecke, H.-C. & Fengler, W. (2012). From Control Design to FPGA Implementation. In Prof. Subhas Chakravarty (Hrsg.), *Technology and Engineering Applications of Simulink* (S. 129–148). InTech.

National Instruments. (2009a). *Multifunction RIO - NI R Series Multifunction* [User Manual]. Zugriff am 08.01.2015 auf http://www.ni.com/pdf/manuals/370489g.pdf

National Instruments. (2009b). *NI PXI-8108* [User Manual]. Zugriff am 08.01.2015 auf http://www.ni.com/pdf/manuals/372561e.pdf

National Instruments. (2014a). *2.26 GHz Intel Core 2 Duo Quad-Core Real-Time Embedded Controller for PXI: NI PXI-8110 RT* [Datenblatt]. Zugriff am 08.01.2015 auf http://www.ni.com/datasheet/pdf/en/ds-310

National Instruments. (2014b). *2.53 GHz Intel Core 2 Duo Real-Time Embedded Controllers for PXI: NI PXI-8108 RT.* Zugriff am 08.01.2015 auf http://sine.ni.com/ds/app/doc/p/id/ds-299/lang/en

National Instruments. (2014c). *NI PXI-7813R.* Zugriff am 08.01.2015 auf http://sine.ni.com/nips/cds/view/p/lang/de/nid/202013

National Instruments. (2014d). *NI PXI-7853R.* Zugriff am 08.01.2015 auf http://sine.ni.com/nips/cds/view/p/lang/de/nid/206323

National Instruments. (2014e). *NI PXI-7854R.* Zugriff am 08.01.2015 auf http://sine.ni.com/nips/cds/view/p/lang/de/nid/206324

National Instruments. (2014f). *Was ist PXI?* [Whitepaper]. Zugriff am 08.01.2015 auf `http://www.ni.com/white-paper/4811/de/`

Negri, L., Sami, M., Macii, D. & Terranegra, A. (2004). FSM-based power modeling of wireless protocols: the case of Bluetooth. In *Low Power Electronics and Design, 2004. ISLPED '04. Proceedings of the 2004 International Symposium on* (S. 369-374). doi: 10.1109/LPE.2004.1349368

Nolte, T., Hansson, H. & Bello, L. (2005). Automotive communications-past, current and future. In *Emerging Technologies and Factory Automation, 2005. ETFA 2005. 10th IEEE Conference on* (Bd. 1, S. 985-992).

Nolte, T., Hansson, H., Norström, C. & Punnekkat, S. (2001). Using bit-stuffing distributions in CAN analysis. In S. L. Iain Bate (Hrsg.), *Proceedings of the IEEE/IEE Real-Time Embedded Systems Workshop in conjunction with the 22nd IEEE Real-Time Systems Symposium (RTSS01).* Technical Report, Department of Computer Science, University of York.

Normen Weller, Johannes Klöckner. (2007). *Transformation von CAN-Bus-Systembeschreibungen vom FIBEX-Format in das MLDesigner-Format mittels XSLT* [Dokumentation].

Nossal, R. (2004). FlexRay - A New Communication Paradigm! A New Development Paradigm? *Hanser Automotive: Electronics systems / Special Edition FlexRay.*

Obermaisser, R. (2012). *Time-triggered communication.* Boca Raton: Taylor & Francis.

OSEK/VDX. (2001a). *OSEK/VDX Fault-Tolerant Communication - Version 1.0* [Specification].

OSEK/VDX. (2001b). *OSEK/VDX Time-Triggered Operating System - Version 1.0* [Specification].

OSEK/VDX. (2004). *OSEK/VDX Communication - Version 3.0.3* [Specification].

OSEK/VDX. (2005). *OSEK/VDX Operating System - Version 2.2.3* [Specification].

Pacholik, A., Klöckner, J., Müller, M., Gushchina, I. & Fengler, W. (2011). LiSARD: LabVIEW Integrated Softcore Architecture for Reconfigurable Devices. In P. M. Athanas, J. Becker & R. Cumplido (Hrsg.), *ReConFig* (S. 442-447). IEEE Computer Society.

Percle, B., Klöckner, J., Manske, E. & Fengler, W. (2011). Signal and data processing in high-precision measuring machines - a case study of NPMM-200. In *Innovation in mechanical engineering - shaping the future - 56th International Scientific Colloquium.* Ilmenau.

Poledna, S., Ettlmayr, W. & Novak, M. (2001). Communication bus for automotive applications. In *Solid-State Circuits Conference, 2001. ESSCIRC 2001. Proceedings of the 27th European* (S. 482-485).

Poledna, S. & Kroiss, G. (1999). TTP : „Drive by Wire“ in greifbarer Nähe. *Elektronik*, *14* , 36–43.

Polke, M. & Ahrens, W. (1994). *Prozeßleittechnik.* Oldenbourg Wissensch.Vlg.

Priese, L. & Wimmel, H. (2008). *Petri-Netze* (2. Aufl.). Springer.

PXI Systems Alliance. (2004). *PXI Hardware Specification - PCI eXtanstions for Instrumentation - Revision 2.2* [Specification].

PXI Systems Alliance. (2005). *PXI Express Hardware Specification - PCI EXPRESS eXtanstions for Instrumentation - Revision 1.0* [Specification].

Rauchhaupt, L. (1994). Performance analysis of CAN based systems. In *1st international CAN Conference.*

Rauh, M. (2009). *Modellierung und Analyse der Leistungsfähigkeit einer Systemarchitektur basierend auf simulativen Untersuchungen* (Bachelorarbeit). Technische Universität Ilmenau.

Rausch, M. (2008). *FlexRay: Grundlagen, Funktionsweise, Anwendung ; 59 Tabellen.* München: Hanser.

Reif, K. (2006). *Automobilelektronik: eine Einführung für Ingenieure : mit 277 Abbildungen und 37 Tabellen.* Vieweg.

Reissenweber, B. (2009). *Feldbussysteme zur industriellen Kommunikation* (3. Aufl.). Oldenbourg Industrieverl.

Riggert, W. (1998). *ATM - Technik und Einführung* (1. Aufl.). bhv Verlag.

Robert Bosch GmbH. (1991). *CAN Specification Version 2.0* [Specification].

Robert Bosch GmbH. (2007). *Autoelektrik/Autoelektronik* (5. Aufl.). Vieweg-Verlag.

Rosello, V., Portilla, J., Krasteva, Y. & Riesgo, T. (2009). Wireless sensor network modular node modeling and simulation with VisualSense. In *Industrial Electronics, 2009. IECON '09. 35th Annual Conference of IEEE* (S. 2685-2689). doi: 10.1109/IECON.2009.5415249

Rupp, C., Queins, S. & Zengler, B. (2007). *UML 2 glasklar: Praxiswissen für die UML-Modellierung* (3. Aufl.). München: Hanser.

Salzwedel, H. (2004). A Framework for Mission Level Design. In *49. Internationales Wissenschaftliches Kolloquium IWK'2004, Ilmenau.*

Schnell, G. & Wiedemann, B. (2006). *Bussysteme in der Automatisierungs- und Prozesstechnik.* Westdeutscher Verlag GmbH.

Schorcht, G., Troxel, I., Farhangian, K., Unger, P., Zinn, D., Mick, C., ... Salzwedel, H. (2003). System-level simulation modeling with MLDesigner. In *Modeling, Analysis and Simulation of Computer Telecommunications Systems, 2003. MASCOTS 2003. 11th IEEE/ACM International Symposium on* (S. 207-212).

Seo, S.-H., Kim, J.-H., Hwang, S.-H., Kwon, K. H. & Jeon, J. W. (2012). A Reliable Gateway for In-vehicle Networks Based on LIN, CAN, and FlexRay. *ACM Trans. Embed. Comput. Syst.*, *11* (1), 7:1–7:24.

Seo, S.-H., Lee, S.-W., Hwang, S.-H. & Jeon, J. W. (2006). Development of Network Gateway Between CAN and FlexRay Protocols For ECU Embedded Systems. In *SICE-ICASE, 2006. International Joint Conference* (S. 2256-2261).

Shaheen, S., Heffernan, D. & Leen, G. (2007). A gateway for time-triggered control networks. *Microprocessors and Microsystems*, *31* (1), 38 - 50.

Showk, A., Szczesny, D., Traboulsi, S., Badr, I., Gonzalez, E. & Bilgic, A. (2009). Modeling LTE Protocol for Mobile Terminals Using a Formal Description Technique. In *Proceedings of the 14th International SDL Conference on Design for Motes and Mobiles* (S. 222–238). Berlin, Heidelberg: Springer-Verlag.

Simon, S. (2010). *Modellierung der Kommunikationsstrukturen von Rechnernetzen der Prozessdatenverarbeitung mit Automatenmodellen und Statecharts* (Bachelorarbeit). Technische Universität Ilmenau.

Somers, B. (2009). *Investigation of a FlexRay - CAN Gateway in the Implementation of Vehicle Speed Control* (Masterthesis). Waterford Institute of Technology.

Stallings, W. (2004). *Data and computer communications* (7. Aufl.). Pearson Edu-

cation.
Sweeney, P. & Höfler, A. (1992). *Codierung zur Fehlererkennung und Fehlerkorrektur.* München: Hanser.
Swoboda, J. (1973). *Codierung zur Fehlerkorrektur und Fehlererkennung.* München: Oldenbourg.
Tanenbaum, A. S. (2009). *Moderne Betriebssysteme* (3., aktualisierte Aufl.). München: Pearson Studium.
Tasak, D. (1997). *Specification and Validation of Q.2931 ATM Signaling Protocol Using Estelle.* McGill University.
Timmler, S. (2011). *Fehlerkorrigierende Codierung für die Datenübertragung - FPGA-basierte Realisierungen für die „NPMM-200“* (Diplomarbeit). Technische Universität Ilmenau.
Tindell, K. & Burns, A. (1994). *Guaranteed message latencies for distributed safety-critical hard real-time control networks* (Bd. 229). Heslington and York: Univ. of York.
Tindell, K. W., Hansson, H. & Wellings, A. J. (1994). Analysing Real-Time Communications: Controller Area Network (CAN). In *Real-Time Systems Symposium, 1994., Proceedings* (S. 259–263).
ttaGroup. (2002). *Time-Triggered Protocol TTP/C - High-Level Specification Document 1.0* [Specification].
TZ Mikroelektronik. (2006). *FLEXENTRY Starter kit for the FlexRay Communication System* [User Manual].
Vector Informatik. (2013). *FlexRay Interface Family* [User Manual].
Vector Informatik GmbH (Hrsg.). (2014). *PressBook - Kompendium ausgewählter Fachartikel zur Elektronik-Entwicklung in verteilten Systemen* (5. Aufl.).
Vonhoegen, H. (2005). *Einstieg in XML.* Galileo Press.
Wallentowitz, H. & Reif, K. (2006). *Handbuch Kraftfahrzeugelektronik: Grundlagen, Komponenten, Systeme, Anwendungen* (1. Aufl.). Wiesbaden: Vieweg.
Wilde, A. (1999). *SDH in der Praxis.* VDE Verlag. (ISBN: 3-8007-2446-4)
Wimmel, H. (2008). *Entscheidbarkeit bei Petri-Netzen.* Springer.
Zacher, S. & Reuter, M. (2011). Regelungstechnik für Ingenieure. In (S. 1-14). Vieweg+Teubner.
Zaghal, R. Y. & Khan, J. I. (2005). *EFSM/SDL modeling of the original tcp standard (RFC793) and the congestion control mechanism of TCP Reno* (Bericht). Kent, Ohio: Kent State University.
Zeigler, B., Praehofer, H. & Kim, T. (2000). *Theory of Modeling and Simulation: Integrating Discrete Event and Continuous Complex Dynamic Systems.* Acad. Press.
Zhu, Z. (2009). *Modellierung des Bussystems Local Interconnect Network (LIN)* (Studienarbeit). Technische Universität Ilmenau.
Zimmermann, W. & Schmidgall, R. (2011). *Bussysteme in der Fahrzeugtechnik: Protokolle, Standards und Softwarearchitektur* (4. Aufl.). Wiesbaden: Vieweg und Teubner.
Zinner, H., Noebauer, J., Gallner, T., Seitz, J. & Waas, T. (2011). Application and realization of gateways between conventional automotive and IP/Ethernet-based networks. In *Design Automation Conference (DAC), 2011 48th ACM/E-*

DAC/IEEE (S. 1-6).

Zschäck, S., Amthor, A., Müller, M., Klöckner, J., Ament, C. & Fengler, W. (2010). Integrated system development process for high-precision motion control systems. In *CCA* (S. 344-350). IEEE.

Zschäck, S., Klöckner, J., Amthor, A., Ament, C. & Fengler, W. (2012). Nanopositionier- und Nanomessmaschinen mittels einer modularen FPGA-Plattform. In Gesch, Niedermaier & Meißner (Hrsg.), *3. Landshuter Symposium für Mikrosystemtechnik* (S. 267-276). Landshut: Hochsch. Landshut.

Zschäck, S., Klöckner, J., Gushchina, I., Amthor, A., Ament, C. & Fengler, W. (2013). Control of nanopositioning and nanomeasuring machines with a modular FPGA based data processing system. *Mechatronics*, *23* (3), 257 - 263. doi: 10.1016/j.mechatronics.2012.12.003